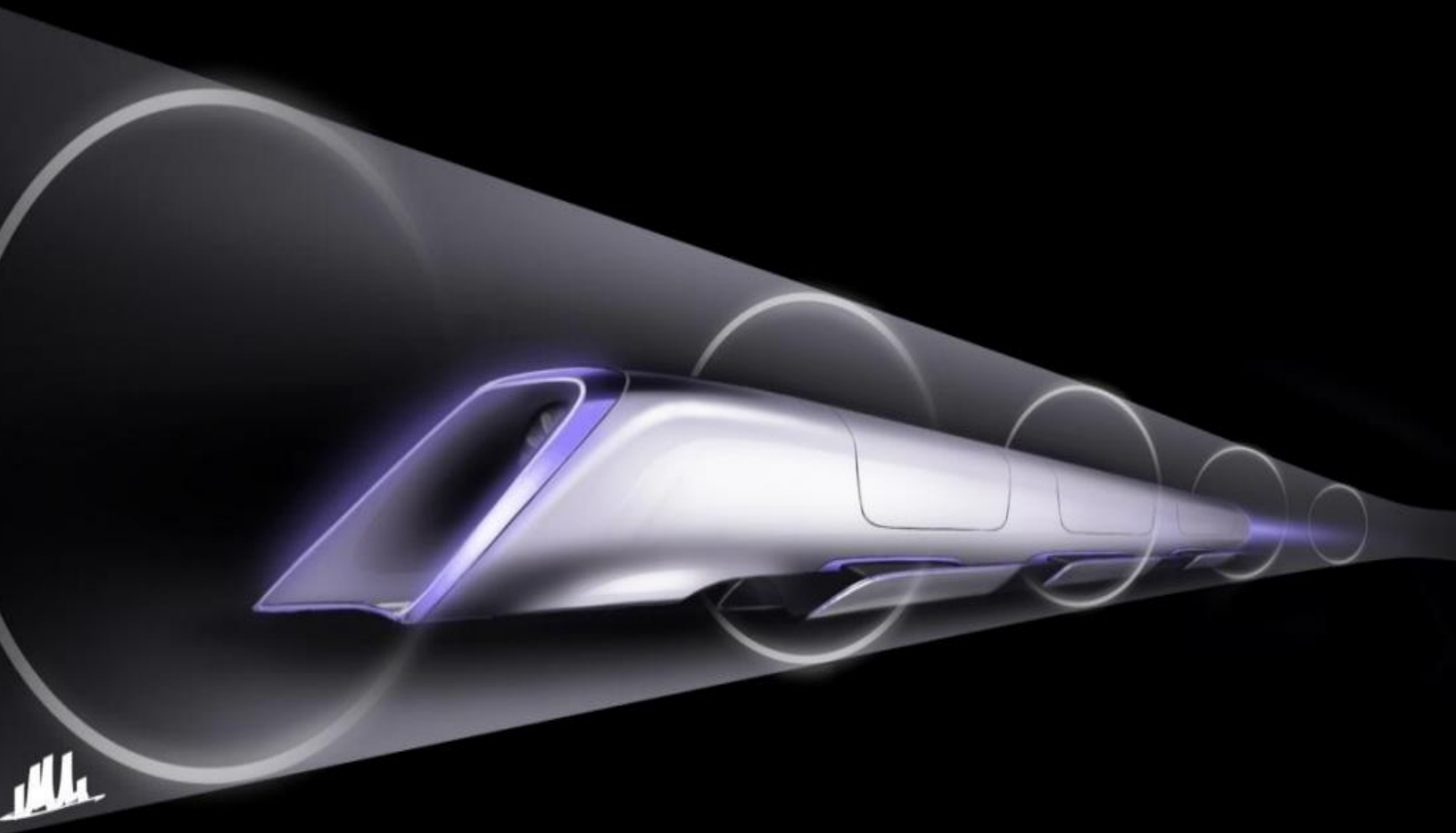


SpaceX Hyperloop Pod
Competition 2016

Nova Hyperloop Team

Final Design Package

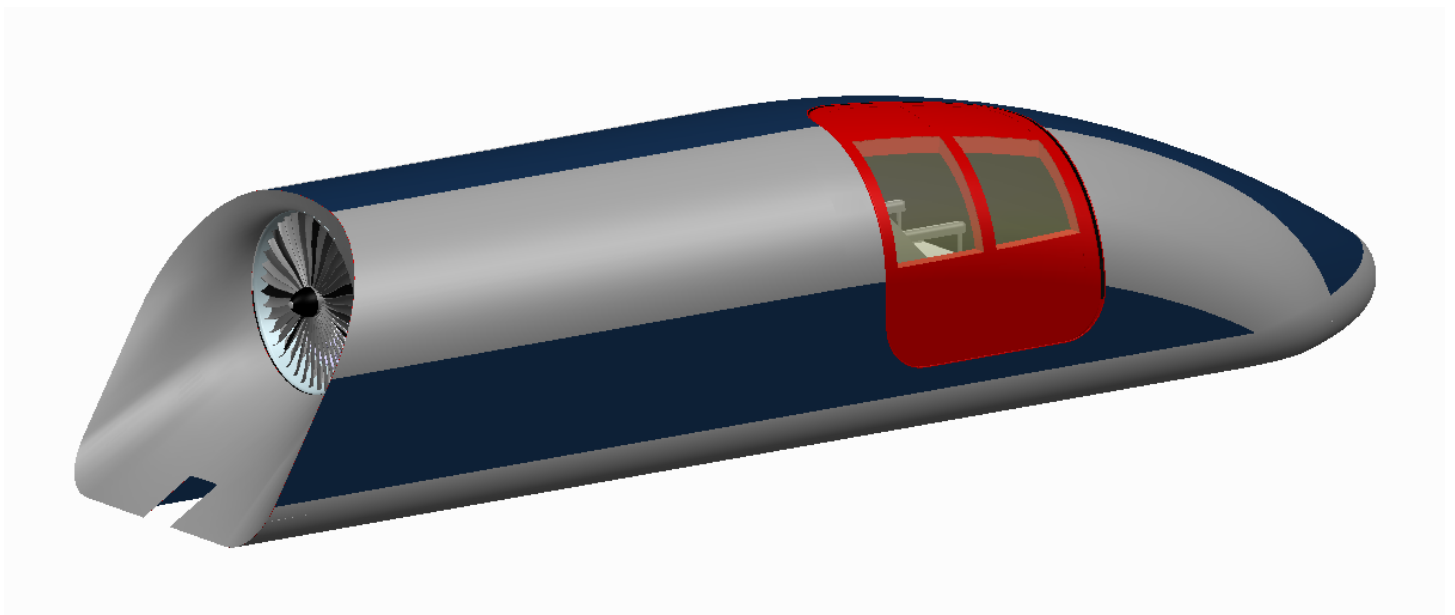


Hyperloop Pod Design

Introduction

On August 12, 2013, Elon Musk released a white paper on the Hyperloop, his concept of high-speed ground transport. In order to accelerate the development of a functional prototype and to encourage student innovation, SpaceX is moving forward with a competition to design and build a half-scale Hyperloop pod.

We found that as a motivation to participate in building such a new and important technology for our future. As Aerospace Engineering students we found that as a chance to learn more about the science that we are already interested in and innovate in building something that never existed before, so we made our goal in the next year to be designing, building and testing the next generation of the future transportation system. It is a great step for our university to be part of SpaceX projects including such a strong competition for a topic that has just been introduced to universe in 2013, so it's time for Egypt through Cairo University to keep up with the world's future technologies.



About the team

As aerospace engineering students, we are so passionate about introducing what's new and innovative to our society. On hearing about SpaceX Hyperloop Pod Competition, we made our goal in the next year be designing, building and testing the next generation of the future transportation system. We challenged ourselves and we intended to do our best to reach our goal. We started our design making it as simple, however efficient, as we can. Our main concern was optimizing every detail in the pod design to get the best performance we can achieve.

Team members:

1. *Dr. Basman Elhadidy (Team advisor)*
2. *Ahmed Mohamed Mohamed
for Navigation, Communication and Control systems*
3. *Mohamed Hassan Mohamed
for Levitation system and Compressor Design*
4. *Samar Fathy Eldiary
for Electrical Power, Braking mechanism, Stability and Safety features*
5. *Ahmed Emad Hodaib
for Propulsion system, Aerodynamic and Structural Design*

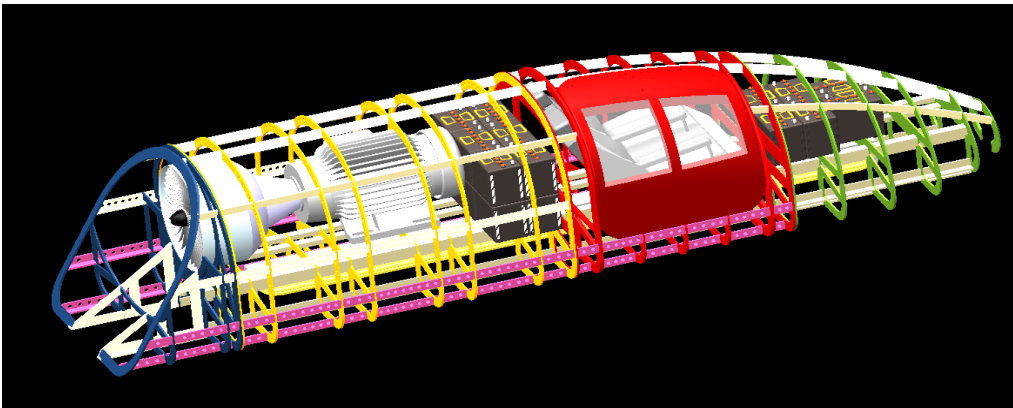
Intention to build the pod

Unfortunately, due to financial issues, the team will not be able to build the pod, the team will not participate in the final phase. However, we wish we can attend the competition weekend.

Overview

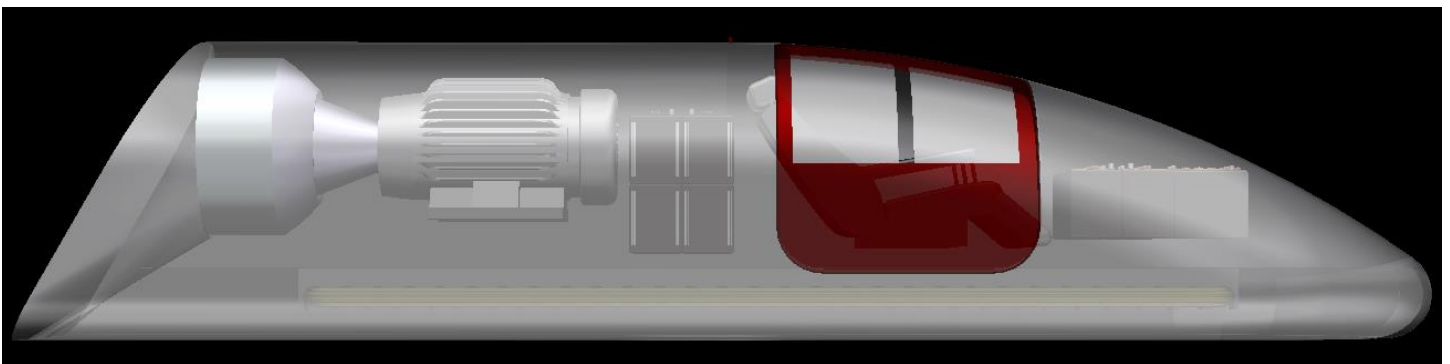
Our pod estimated weight is 1500 kg and its subsystems are mainly:

- *Single-sided linear induction motor for propulsion*
- *Air-bearing-slider mechanism for levitation, using a compressor and a an electric motor*
- *DC batteries supplying power to the pod, It is converted to 3-phase high-frequency AC for propulsion*
- *A structural design using steel as a material*
- *A GPS-aided inertial navigation system*



To increase stability of the pod, we iterated for the final configuration until the CG of the pod is located at the optimized location for different scenarios.

The batteries (340 kg) are placed in the rear part of the pod and some are ahead of the seat part, while the linear induction is place in the bottom of the pod. The Compressor and its motor are placed in the front.



Propulsion System

Abstract

To achieve the high target velocity of the pod, we found that an electromagnetic system is the most efficient and clean way to generate the required contactless thrust force. Hence, we decided to use Linear Induction Motor “LIM” as our propulsion system. In spite of its few disadvantages, its maintenance is very easy as it is free of gearboxes and moving parts, moreover, its cost is much less than that of the rotary electric motors.

As with rotary motors, linear motors frequently run on a three-phase power supply and can support very high speeds. However, there are end-effects that reduce the motor's thrust force. Linear induction motors are thus less energy efficient than normal rotary motors for any given required force output. These end effects include losses in performance and efficiency that are believed to be caused by magnetic energy being carried away and lost at the end of the primary by the relative movement of the primary and secondary. One other parameter that affects the efficiency of the motor and increases the power loss is the air gap between the rotor and the stator of the LIM.

A linear induction motor consists of two parts: The primary part consists of a laminated iron core “thin insulated sheets” with transverse slots that are often straight cut with coils into the slots, with each phase giving an alternating polarity so that the different phases physically overlap. The secondary part is frequently a sheet of aluminium. Some LIMs are double-sided with one primary on each side of the secondary and some are single-sided.

The thrust force is produced by a linearly moving magnetic field acting on conductors in the field. This conductor, that is placed in this field, will have eddy currents induced in it thus creating an opposing magnetic field in accordance with Lenz's law. The two opposing fields will repel each other, creating motion as the magnetic field sweeps through the metal.

A linear induction motor may be a short primary, where the primary part is the moving part “the rotor” and the secondary part is the stator, also it may be a short secondary, where the conductive plate is truncated smaller. Generally, the secondary is made wider than the primary to make maximum use of the primary magnetic field.

The winding configuration is very important in the design of linear induction motor. After several design iterations, we ended up with a double-layer-winding full-pitch linear induction motor. Double-winding means that each slot contains two coil sides. Full-pitch means that the coil span is 180 degrees “one pole pitch”.

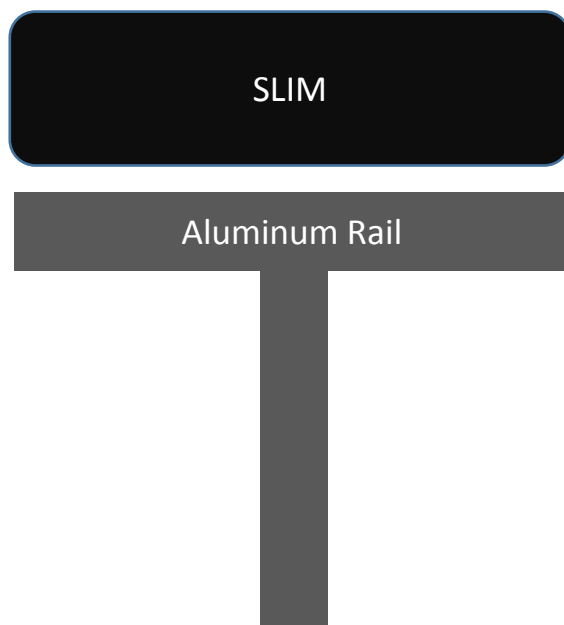
Our motor is short primary, to be compatible with the test track. We found that using a single-sided LIM will minimize the end-effects and also the air gap, as we can put the primary part right over the aluminum I-channel of the rail.

Slipping is the relative motion between the traveling magnetic field and the stator. It is needed to induce voltage in the conductor.

We designed our SLIM for a slipping rate of 10% and an air gap of 1 cm. However, The motor works properly in a wide range of slipping rate and air gap around these values, but with a slight decrease in the efficiency.

During our design, we faced some problems concerning the power source specifications, we tried to make our design compatible with the available high-frequency and high-voltage 3-phase power sources. After optimizing the power requirements for the target thrust force and velocity and minimizing the power consumption of the motor by reducing the end effects, we tried to minimize the weight of the motor. We were restricted by the maximum allowable flux density “B” in the different parts of the iron core. Besides, we were restricted by the maximum allowed current density “J” in the wires of the coil turns, as well.

We started to study the different scenarios of motion, we studied the variation of the thrust force with the velocity, air gap and the slipping rate, as well. After many iterations, we finally got satisfied with our final results, our designed LIM showed a satisfying goodness factor.



List of symbols

As	Area of slot
Aw	Area of copper wire for one turn per slot
Bgavg	Average air-gap magnetic flux density
Bgmax	Maximum air-gap magnetic flux density
Btmax	Maximum tooth flux density
Bymax	Maximum yoke flux density
d	Thickness of the aluminum sheet
Dw	Diameter of Copper wire
e	Induced voltage per turn in a coil
E1	RMS Induced Voltage in a coil
f	Source frequency
Fs	Electromagnetic thrust generated by rotor
ge	Equivalent air-gap
gm	Mechanical air-gap
g0	Magnetic air-gap
G	Goodness factor
hs	Slot height
hy	Yoke Height
I1	RMS Input Phase Current
I2	Stator Phase Current
Im	Magnetizing current
J1	Rotor Current Density
Jm	Rotor Current Density
kc	Carters Coefficient

kd	Distribution factor
kp	Pitch factor
kw	Winding factor
lw	Length of copper wire
Ls	Length of one Rotor unit
m	Number of Phases
N1	Turns per phase
Nc	Number of Turns per Slot
Np	Number of parallel wires
p	Number of Poles
Pi	Input Power
Po	Output Power
q1	slots per pole per phase
R1	Per phase Rotor Resistance
R2	Per phase Stator Resistance
S	Slip
V1	RMS Input Phase Voltage
Vline	Source line voltage
Vr	Rotor Velocity
Vs	Synchronous velocity
ws	slot width
wt	Tooth width
Wtmin	Minimum Tooth Width
Ws	Width of the core
Wse	Equivalent core Width
X1	Per-phase Rotor-slot Leakage Reactance

X_m	Per-phase Magnetizing Reactance
Z	Unit Impedance
ϕ	Angle between voltage and current
α	Slot angle
η	Efficiency
λ	Slot Pitch
μ_0	Permeability of free space
θ_p	Coil span
ρ_r	Volume resistivity of Aluminum
ρ_w	Volume resistivity of Copper
τ	Pole pitch
Φ	Flux in a coil
Φ_p	Flux Linkage per pole

Design calculations

- **Current Sheet Concept:** To study the MMF produced by the coils, we assume a thin layer of current that produces the same sine wave MMF in the air gap as the coils.

1. Assigning the constant values to μ_0 , ρ_r , ρ_w , J_1 , B_{tmax} and B_{ymax}

$$\mu_0 = 4\pi * 10^{-7} \text{ H/m}$$

$$\rho_r = 19.27 * 10^{-9} \text{ } \Omega\text{-m}$$

$$\rho_w = 28.85 * 10^{-9} \text{ } \Omega\text{-m}$$

$$J_1 = 6 * 10^6 \text{ A/m}^2$$

$$B_{tmax} = 1.6 \text{ T}$$

$$B_{ymax} = 1.3 \text{ T}$$

2. Specifying desired values for m , q_1 , p , S , W_s , g_m , d , V_{line} , f , V_r and F_s

$$m = 3 \text{ phases}$$

$$q_1 = 1 \text{ slots/pole/phase}$$

$$p = 10 \text{ poles}$$

$$S = 10\%$$

$$W_s = 12.5 \text{ cm}$$

$$g_m = 1 \text{ cm}$$

$$d = 1.046 \text{ cm}$$

$$V_{line} = 960 \text{ V}$$

$$f = 500 \text{ HZ}$$

$$V_r = 285 \text{ m/s}$$

$$F_s = 545 \text{ N}$$

3. V_s can be calculated from V_r and S

$$V_s = \frac{V_r}{1 - S}$$

4. V_1 is computed from V_{line}

$$V_1 = \frac{V_{line}}{\sqrt{3}}$$

5. τ , λ , L_s are calculated

$$\tau = \frac{V_s}{2f}$$

$$\lambda = \frac{\tau}{m * q_1}$$

$$L_s = p * \tau$$

6. Assuming $w_s = w_t = \lambda/2$ and putting $N_c=1$

7. N_1 is computed

$$N_1 = p * q_1 * N_c$$

8. Initially guessing a value for $\eta \cos \varphi$ where $0 < \eta \cos \varphi < 1$

9. I_1 is calculated

$$I_1 = \frac{F_s * V_r}{m * V_1 * \eta \cos \varphi}$$

10. A_w , A_s and h_s are assumed, assuming 30% of the area of the slot is filled with insulation material

$$A_w = \frac{I_1}{j_1}$$

$$A_s = \frac{10}{7} N_c * A_w$$

$$h_s = \frac{A_s}{w_s}$$

11. g_o , γ , k_c , g_e and G are calculated

$$g_o = g_m + d$$

$$\gamma = \left(\frac{4}{\pi}\right) * \left[\left(\frac{w_s}{2g_o} * \tan^{-1}\left(\frac{w_s}{2g_o}\right)\right) - \log\left(\sqrt{1 + \left(\frac{w_s}{2g_o}\right)^2}\right)\right]$$

$$k_c = \frac{\lambda}{\lambda - \gamma * g_o}$$

$$g_e = k_c * g_o$$

$$k_w = \frac{\sin\left(\frac{\pi}{2 * m}\right)}{q_1 * \sin\left(\frac{p_i}{2 * m * q_1}\right)}$$

$$G = 2 * \mu_0 * f * \frac{\tau^2}{\pi * \left(\frac{\rho r}{d}\right) * g_e}$$

12. Per-phase equivalent circuit components X_1 , R_1 , X_m and R_2 are calculated
13. Z , φ , $\cos\varphi$, I and I_m are calculated
14. At V_r , we can calculate the actual F_s , P_o , P_i and η
15. $\cos\varphi$ is calculated
16. If the calculated $\eta\cos\varphi$ error from the assumed value is less than 1% then proceed to the next step, If not, we repeat from step 6 with a new value of $\cos\varphi$

$$\cos\varphi(\text{new}) = \frac{\cos\varphi(\text{calculated}) + \cos\varphi(\text{old})}{2}$$

17. Then checking if the calculated thrust force equals the target thrust force. If not, we increase N_c by 1 and repeat from step 4
18. Put $N_p=1$
19. Choosing Copper a wire size and arrangement
20. Calculating the new values of w_s , w_t and h_s
21. g_o , γ , k_c , g_e and G are calculated again using the same equations
22. New X_1 , R_1 , X_m and R_2 are calculated
23. W_{tmin} is calculated
24. Checking if $w_t > w_{tmin}$. If not, we increment N_p and repeat from step 17
25. Calculating the new I_1 and the actual J_1
26. Calculating the minimum h_y
27. At rated V_r , we can calculate F_s , P_o , P_i and η
28. Finally, we check if the actual thrust force equals the target thrust force. If not, we choose a new wire gauge and repeat from step 18

Winding configuration

Phase A

Pattern: In(1)(1-4)(4-7)(7-10)(10-13)(13-16)(16-19)(19-22)(22-25)(25-28)(28-25)(25-22)(22-19)(19-16)(16-13)(13-10)(10-7)(7-4)(4)Out
-Enters at slot 1 and exits at slot 4

Phase B

Pattern: In(3)(3-6)(6-9)(9-12)(12-15)(15-18)(18-21)(21-24)(24-27)(27-30)(30-27)(27-24)(24-21)(21-18)(18-15)(15-12)(12-9)(9-6)(6)Out
-Enters at slot 3 and exits at slot 6

Phase C

Pattern: In(2)(2-5)(5-8)(8-11)(11-14)(14-17)(17-20)(20-23)(23-26)(26-29)(29-26)(26-23)(23-20)(20-17)(17-14)(14-11)(11-8)(8-5)(5)Out
-Enters at slot 2 and exits at slot 5

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
A	C	B	A'	C'	B'	A	C	B	A'	C'	B'	A	C	B	A'	C'	B'	A	C	B	A'	C'	B'	A	C	B			
			A'	C'	B'	A	C	B	A'	C'	B'	A	C	B	A'	C'	B'	A	C	B	A'	C'	B'	A	C	B	A'	C'	B'

Coil span $\vartheta_p = 180^\circ$ (Full-pitch)

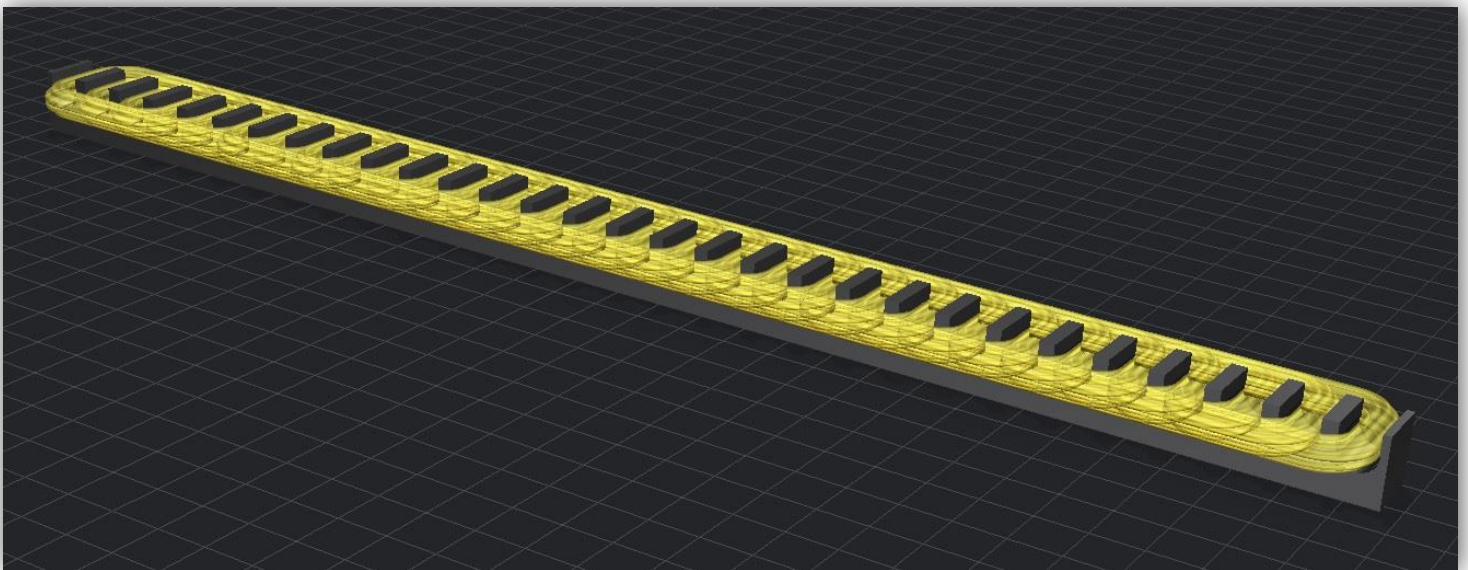
Double-layer winding

Number of poles $p = 10$

Number of slots = 30

Number of phases $m = 3$

Slots per pole per phase $q_1 = 1$



Results

Number of phases = 3

Number of poles = 10

Number of slots per pole per phase = 1

Coil Span = 180°

Rotor Velocity = 285 m/s

Slip Percentage = 10%

Frequency = 500 HZ

Line Voltage = 960 V

Aluminum Sheet thickness = 1.046 cm

Air gap = 1 cm

Wire guage = 8

Wire diameter = 3.2639 mm

Number of turns per coil = 2

Parallel wires per winding = 27

Length of core = 3.1667 m

Width of core = 12.5 cm

Slot width = 9.0325 cm

Tooth width = 1.5230 cm

Minimum tooth width = 0.0203 cm

Minimum Yoke height = 0.0476 cm

Minimum Slot height = 0.7146 cm

Slot pitch = 10.5556 cm

Minimum Slot Area = 6.4545 cm²

Rated Current = 658.3745 A

Current density = 2.9144e+06 A/m²

Thrust Force = 544.9501 N

Output Power = 208.4708 HP

Input Power = 231.6881 HP

Efficiency = 0.8998

Weight estimation

$$W_{total} = W_{wire} + W_{core}$$

$$\text{Mass density of iron} = 8000 \text{ Kg/m}^3$$

$$\text{Mass density of copper} = 8950 \text{ Kg/m}^3$$

$$W_{wire} = \text{density of copper} * \text{wire cross-sectional area} * \text{number of turns} * \text{no. of slots} * 2 (\text{Slot pitch} + \text{core width}) = 27 \text{ kg}$$

$$W_{core} = \text{density of iron} * \text{core width} * (\text{yoke height} * \text{core length} + \text{tooth height} * \text{tooth width} * \text{no. of slots}) = 180 \text{ kg}$$

Then, the estimated total weight of the LIM is 207 kg

Comment on Production

The manufacturing of the linear induction motor rotor is very easy, as there is no moving complicated parts. The core is made of steel laminated, insulated using resin, and put together to form the laminated core. This can be made by using CNC laser cutting or water jet cutting. The core side is drawn on a CAD software and exported as a 2D drawing to be ready for cutting. The copper coil turns are also insulated from each other using resin. Some threaded holes are made in specific parts of the core to be fastened to the structure of the pod using bolts.

Cost

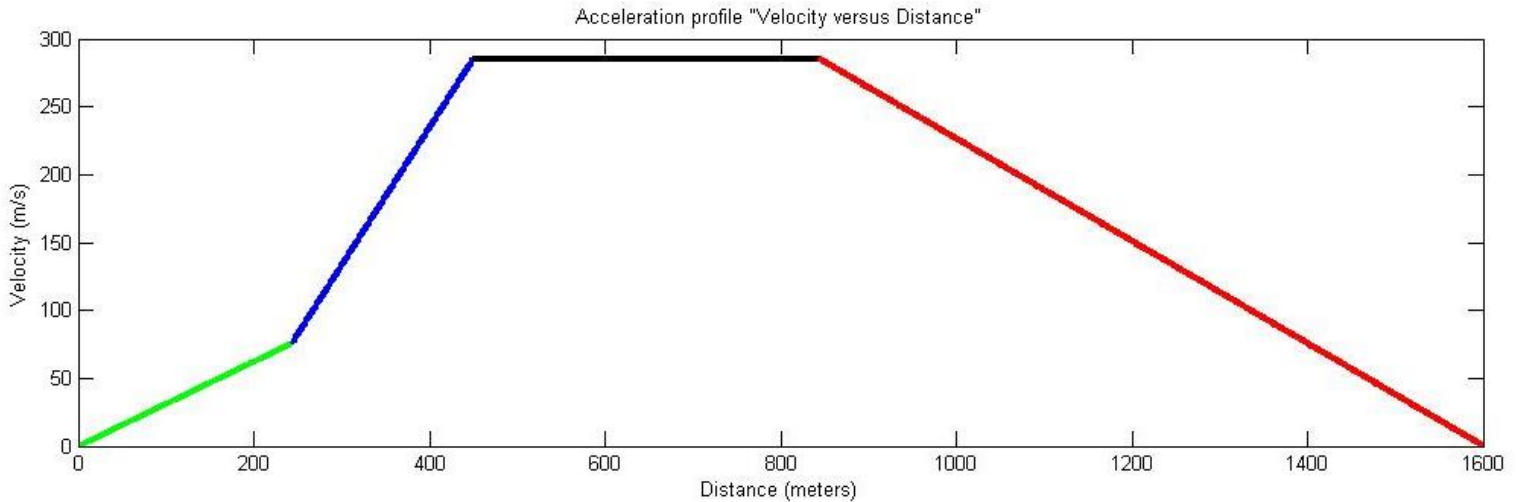
The manufacturing is very cheap as the manufacturing is so easy. However, with the estimated cost for the components, such as steel sheets and copper wires, the cost becomes

$$\text{Total cost} = 3800 + 1500 + 2400 = 7,700\$$$

Maintenance

Since the linear induction motor has no moving parts, no corrosion and no wearing takes place, so there is no need for lubrication. The coils are easy to change if it is cut damaged. The core laminas can easily be replaced if needed. However, the linear induction motor does not require a regular maintenance.

Pod trajectory



The green and blue lines represent the change of velocity with distance for the acceleration phase. The pod accelerates with 1.2g with the help of a pusher. Then acceleration increases with the increase of the linear induction motor frequency, till it reaches the required velocity of the pod. The red line represents the braking phase, where the deceleration is about -5g.

Matlab code

```
% SLIM Design Using a Matlab Program
clear all;
close all;
clc;

%Inputs
no_of_phases = 3 ; %Number of phases
Vline = 960; %Line Voltage
f = 500; %Frequency
p = 10; %Number of poles
q1= 1; %Number of slots per pole per phase
Srate = 0.1; %Rate of Slip
Ws = 0.125; %Width of Core
gm = 0.01; %Air Gap in meters
d = 0.01046; %Aluminum Sheet thickness in meters
Fsprime = 900; %Target Thrust Force
Vrated= 285; %Rated Velocity mps

%Assign necessary constants and parameters
mu0 = 4*pi*10^-7;
rho_w = 19.27*10^-9;
rho_r = 28.85*10^-9;
btmax = 1.6;
bymax = 1.3;
Jl = 6e6;
```



```

V1 = Vline/sqrt(3);
Vs= Vrated/(1 - Srate);
tau = Vs/(2*f);
lambda = tau/(no_of_phases*q1);
Ls = p*tau;

for i = 1:30
N1 = p*q1*i;
ncos0 = 0.2;
ncos1(i) = 1;

while abs(ncos0 - ncos1(i))>0.0001
I1prime = (Fsprime*Vrated)/(no_of_phases*V1*ncos0);
Aw = I1prime/J1;
As = (10*i*Aw)/7;
ws = lambda/2;
wt = ws;
hs = As/ws;
go = gm + d;
gamma = (4/pi)*(((ws/(2*go))*atan(ws/(2*go))) - log(sqrt(1 + ((ws/(2*go))^2)))));
kc = lambda/(lambda - gamma*go);
ge = kc*go;
kw = sin(pi/(2*no_of_phases))/(q1*sin(pi/(2*no_of_phases*q1)));
G = 2*mu0*f*tau^2/(pi*(rho_r/d)*ge);
a=pi/2;
ae=a+ge/2;
Lce=tau;
beta1=1;
lamda_s= (hs*(1+3*beta1))/(12*ws);
lamda_e= (0.3*(3*beta1-1));
lamda_d= 5*(ge/ws)/(5+4*(go/ws));

%Equivalent Circuit Components
R1(i)=rho_w*(4*a+2*Lce)*J1*N1/I1prime;
a1(i)=lamda_s*(1+3/p)+lamda_d;
b1(i)=lamda_e*Lce;
X1(i)=8*mu0*pi*f*((a1(i)*2*a/q1)+b1(i))*N1^2/p;
Xm(i)= (48*mu0*pi*f*ae*kw*N1^2*tau)/(pi^2*p*ge);
R2(i) = Xm(i)/G;
Z(i)=R1(i)+j*X1(i)+((j*R2(i)*Xm(i))/Srate)/((R2(i)/Srate) + j*Xm(i));
I1(i) = V1/abs(Z(i));
I2(i) = j*I1(i)*Xm(i)/(R2(i)/Srate+j*Xm(i));
Im(i) = I1(i) - I2(i);

%Actual TLIM Thrust
Fs(i) = (no_of_phases*abs(I1(i))^2*R2(i))/(((1/(Srate*G))^2)+1)*Vs*Srate);
diff(i) = Fs(i) - Fsprime;
dmin = min(abs(diff));
Pout = Fs*Vrated;
Pin=Pout+no_of_phases*abs(I2(i))^2*R2(i)+no_of_phases*abs(I1(i))^2*R1(i);
eta = Pout/Pin;
PF = cos(angle(Z(i)));
ncos1(i)=eta*PF;
ncos0=(ncos0+ncos1(i))/2;

end;
end;

k = 1;
while dmin~=abs(diff(k))
k = k + 1;

```

```

end;
Nc = k;
N1 = p*q1*Nc;
Fs = Fs(k);
I1 = I1(k);
ncos1 = ncos1(k);

A=[2 6.543;3 5.8;4 5.189;5 4.62;6 4.1148;7 3.665;8 3.2639];
guage=0;
while (guage<7)
guage=guage+1;
pw = 0;
r=0;
wt = 1;
wtmin= 0;
g=0;r=0;
while (wt-wtmin)>0.0152
r=r+1;
g=g+1;
wire_d=A(guage,2);
pw = pw + 1;
ws = (wire_d*10^-3*pw) + 2.2*10^-3;
wt = lambda - ws;
Aw = pw*pi/4*wire_d^2*1e-6;
As = (10*Nc*Aw)/7;
hs = As/ws;
go = gm + d;
gamma=(4/pi)*(((ws/(2*go))*atan(ws/(2*go)))-log(sqrt(1 + ((ws/(2*go))^2)))));
kc = lambda/(lambda - gamma*go);
ge = kc*go;
G = 2*mu0*f*tau^2/(pi*(rho_r/d)*ge);
kw=sin(pi/(2*no_of_phases))/(q1*sin(pi/(2*no_of_phases*q1)));
a=pi/2;
ae=a+ge/2;
Lce=tau;
beta1=1;
lamda_s= (hs*(1+3*beta1))/(12*ws);
lamda_e= (0.3*(3*beta1-1));
lamda_d= 5*(ge/ws)/(5+4*(go/ws));
%Equivalent Circuit Components
R1=rhow*(4*a+2*Lce)*J1*N1/I1prime;
a1=lamda_s*(1+3/p)+lamda_d;
b1=lamda_e*Lce;
X1=8*mu0*pi*f*((a1^2*a/q1)+b1)*N1^2/p;
Xm=(48*mu0*pi*f*ae*kw*N1^2*tau)/(pi^2*p*ge);
R2 = Xm/G;
Z=R1+j*X1+(R2/Srate*j*Xm)/(R2/Srate+j*Xm);
I1 = V1/abs(Z);
I2 = j*I1*Xm/(R2/Srate+j*Xm);
Im=I1-I2;
wtmin=2*sqrt(2)*no_of_phases*kw*N1*abs(Im)*mu0*lambda/(pi*p*ge*btmax);
end;
hy=4*sqrt(2)*no_of_phases*kw*N1*abs(Im)*mu0*ls/(pi*pi*p*p*ge*bymax);
para_wires(guage)=pw;
slot_width(guage)=ws;

tooth_width(guage)=wt;
min_toothwidth(guage)=wtmin;
height_slot(guage)=hs;
Area_wire(guage)=Aw;
Area_slot(guage)=As;
Num_c(guage)=Nc;
Num_1(guage)=N1;

```

```

Sta_I(guage)=I1;
gap_e(guage)=ge;
current_den(guage) = abs(I1)/Aw;
height_yoke(guage)=4*sqrt(2)*no_of_phases*kw*N1*abs(Im)*mu0*Lv/(pi*pi*p*p*ge*bymax);
final_thrust(guage)=(no_of_phases*abs(I1)^2*R2)/(((1/(Srate*G)^2)+1)*Vs*Srate);
output(guage)=final_thrust(guage)*Vrated;
input(guage)=output(guage)+no_of_phases*abs(I2)^2*R2+no_of_phases*abs(I1)^2*R1;
efficiency(guage)= output(guage)/input(guage);
difference(guage)=final_thrust(guage)-Fsprime;
diffmin(guage) = min(abs(difference));
end;

```

```

kk = min(diffmin);
jj=1;
while kk~=abs(diffmin(jj))
jj = jj + 1;
end;

```

%Outputs

```

Number_of_phases=no_of_phases
Number_of_poles=p
Number_of_slots_per_pole_per_phase=q1
Coil_Span_degrees=180
Rated_Velocity_mps=Vrated
Slip_Percentage=Srate*100
Frequency_Hz=f
Line_Voltage=Vline
Aluminum_Sheet_thickness_cm=d*100
Air_gap_cm= gm*100
wire_guage=A(jj,1)
gauge=jj;
Wire_diameter_mm=A(jj,2)
Number_of_turns_per_coil=Num_c(guage)
Number_of_turns_per_phase=Num_1(guage);
Parallel_wires_per_winding=para_wires(guage)
Length_of_core_meters=Ls
Width_of_core_meters=Ws
Slot_width_cm=slot_width(guage)*100
Tooth_width_cm=tooth_width(guage)*100
Minimum_tooth_width_cm=min_toothwidth(guage)*100
Minimum_Yoke_height_cm=height_yoke(guage)*100
Minimum_Slot_height_cm=height_slot(guage)*100
Aw=Area_wire(guage);
Slot_pitch_cm=lambda*100
Minumum_Slot_Area_cm2=Area_slot(guage)*10000
Rated_Current_Ampere=Sta_I(guage)
ge=gap_e(guage);
Current_density=current_den(guage)
Thrust_Force_Newtons=final_thrust(guage)
Output_Power_hp=output(guage)/745
Input_Power_hp=input(guage)/745
Efficiency=efficiency(guage)

```

```

$$$$ To Generate the Characteristic curves $$$
vel_sta= 285;

```

```

slip= 0.1;
e=1;
for slip=0.000001:0.01:1
vel_rot(e)=vel_sta*(1-slip);

```

```

impz(e) = R1+j*Xl+(R2/slip*j*Xm)/(R2/slip+j*Xm);
il(e) = V1/abs(impz(e));
i2(e) = j*i1(e)*Xm/(R2/slip+j*Xm);
im(e) = i1(e)-i2(e);
Force(e)=(no_of_phases*(abs(i1(e)))^2*R2)/(((1/(slip*G)^2)+1)*vel_sta*slip);
out_pow(e) = Force(e)*vel_rot(e);
in_pow(e)=out_pow(e)+no_of_phases*abs(i2(e))^2*R2+no_of_phases*abs(i1(e))^2*R1;
eff(e) = out_pow(e)/in_pow(e);
e=e+1;
end;

%figure(1);
%plot(vel_rot,Force,'green');
%hold on;
%plot([285 285],[0,Fs]);
%hold on;
%plot([0 285],[Fs Fs]);
%hold on;
%figure(2);
%plot(vel_rot,eff*100,'green');
%hold on;
%plot([285 285],[0 eta*100]);
%hold on;
%plot([0 285],[eta*100,eta*100]);
%hold on;

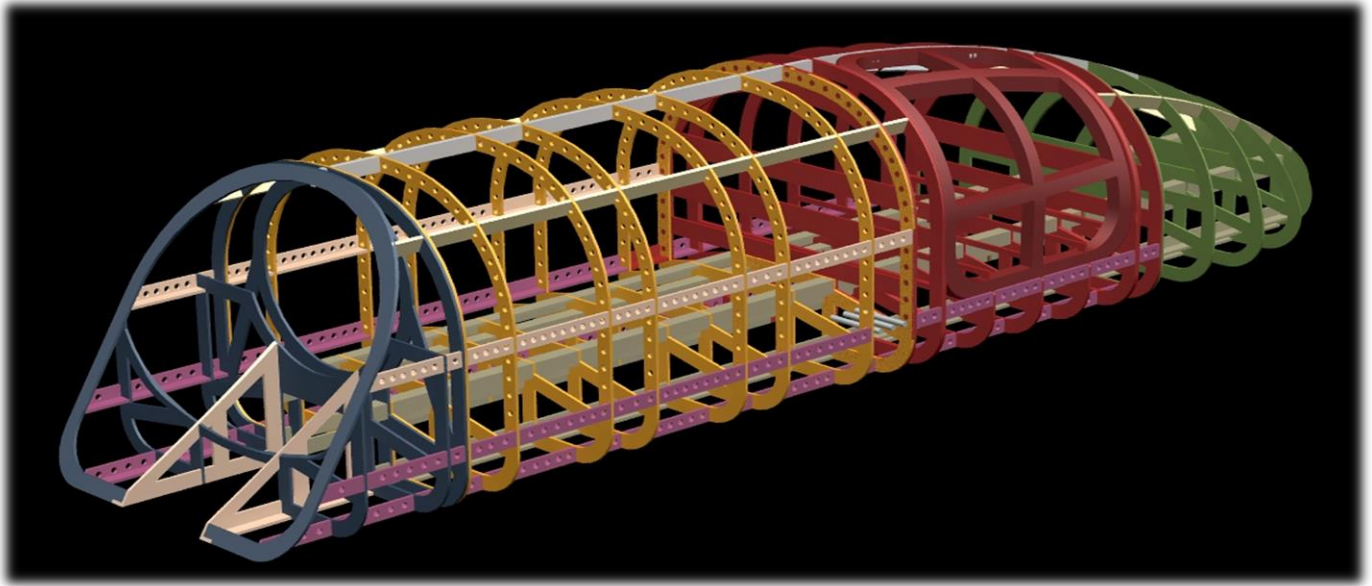
```

References

- [1] Nasar, S.A. and Boldea, I., *Linear Electric Motors*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1987.
- [2] "Electrical traction independent of adhesion," *Le Genie Civil*, pp. 381-382, 1901.
- [3] Zehden, A., "Travelling wave electric traction equipment," French patent 321 692, applied for June 2, 1902.
- [4] Zelenay, Rosenfeld, and Dulait, "Travelling wave applied to electric railways," French patent 318 634, Feb. 12, 1902.

Structural Design

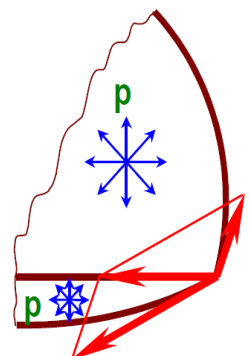
During our structural design, Our main concern was to reduce the weight without affecting the safety. Besides, we tried to minimize the manufacturing cost and make the manufacturing as simple as possible, as well. We tried to make use of the maximum capacity in the pod.



Our structural design was inspired by the airplane fuselage structure. We used transverse frames held apart by longeron channels and stringers. We used flanges to join the different parts of the structure by bolts. The stressed skin shells are fixed to flanges by rivets. We used panel stiffeners in some regions in the pod to overcome the bending loads created by the transverse pressure on the skin. Local reinforcements are finally integrated in some areas such as the door cut-out.

Our design was mainly based on the worst case, however, we studied the different structural cases each alone. We placed the different parts such that the division of forces and the dynamic load transfer become acceptable. We avoided sharp edging and we considered the stress concentration.

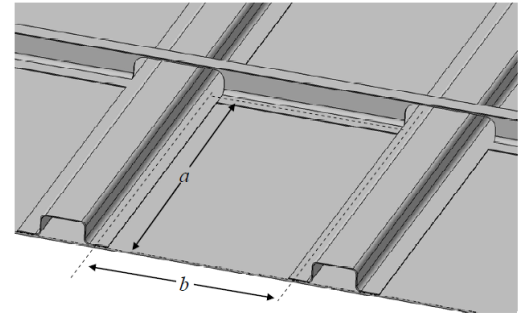
The main loads on the pod are determined to be mainly: Axial loads due to thrust acceleration and braking deceleration, Transverse loads due to pressure acting on the skin, Weight distributed loads of the different subsystems, Bending moments due to different weight distribution and pressure coefficients on the pod different regions and



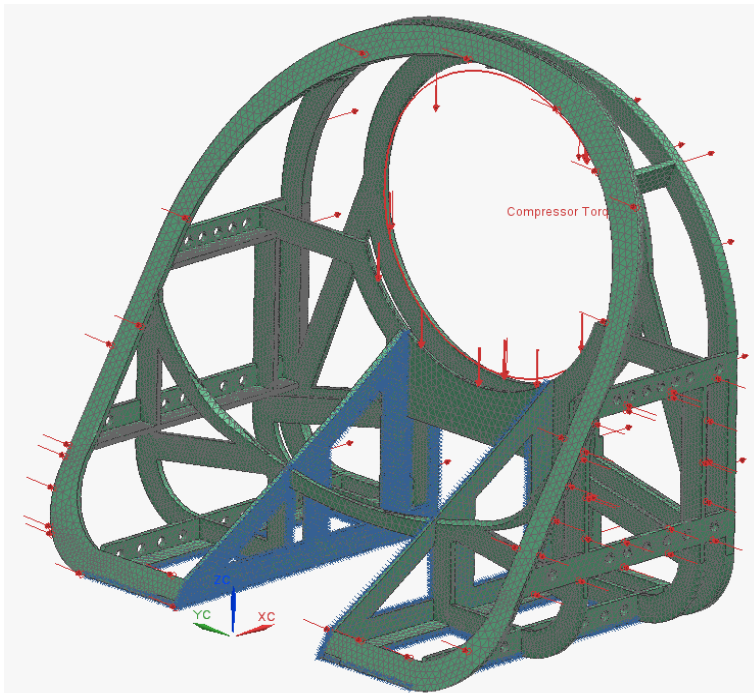
finally torsional stresses in the parts holding the compressor. We studied the thermal stresses on the pod. Finally, choosing the main material to be mainly Steel (Grade 60) for its good thermal properties beside the mechanical properties, and the skin material to be Aluminum (Al-7075) for its low surface roughness and high strength-to-weight ratio, we found that all the results are acceptable for a good margin of safety and all the stresses are within the allowable range.

We were able to reduce the weight as we reduced the joining areas by the integration of different structure parts. Moreover, we avoided the flat pressurized panels and the non-stresses parts. One other advantage of reducing the joining areas is minimizing the bearing stresses which is difficult to overcome.

We used structural concepts in our design such as cross-section idealization. However, we used the **Finite Element Method (FEM)** for our final design results. By determining the overall load on each part and expecting the dynamic loads for different cases, we could analyze the stresses in the part by meshing and using an iterative solver.



Our structure consists of 5 parts: The compressor structure, the middle structure, the seat part, the linear induction motor holding beams and the rear structure. Each consists of bulkheads and longerons.



For, Initial acceleration case ($a = 1.2g$), The simulated structure is safe is stable, The maximum stress was in the region around the compressor ant it was within the allowable range with a safety factor of 4.

For the Nominal deceleration case, the braking loads was added to the simulation, after adding some reinforcements and iterating, the structure finally showed stability.

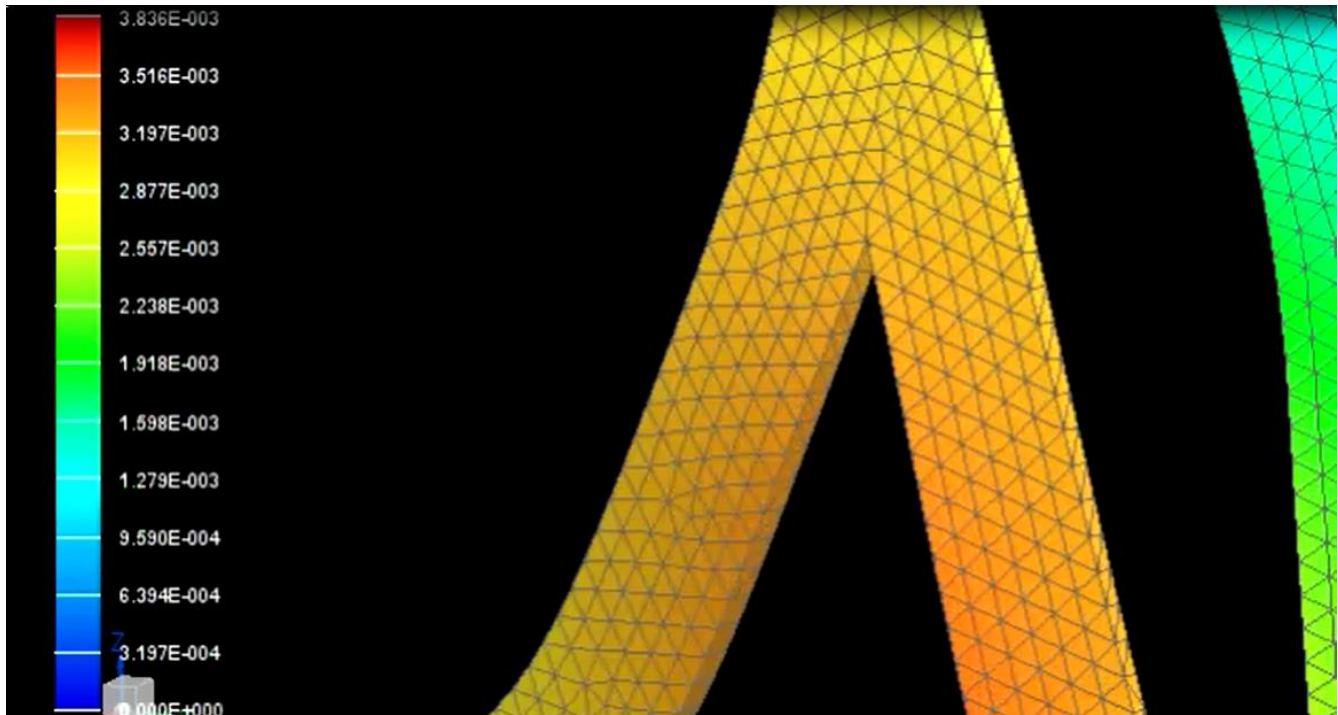
We also studied the effect impact loads for the case of tube end crashing.

For the worst case, the pod goes down a slope of 45 degrees with the maximum braking forces

and maximum compressor torsional loads.

We studied the vibrations effect, specially around the compressor. This is shown in the video.

We studied the overall stiffness of the idealized structure. Finally, Other than the pod subsystems loads, Our structure is capable of transporting a payload of 150 kg safely.



Sample of a part structural analysis result

Specifications

Weight = 650 kg

Dimensions: 110 cm in height and 125 cm in width

Length = 5 meters

Maximum payload = 150 kg

Overall Safety Factor = 2.5

CG position (x, y, z) = (2.362, 0.035, 0.362) m

Centroidal Moments of Inertia (I_{xc}, I_{yc}, I_{zc}) = (223.865, 1405.071, 1434.592) Kg.m²

Where x, y, z are the pod main axes with the origin at the centre of leading edge.

X is the axial axis (+ve backwards), Y is the lateral axis (+ve right) and Z is the vertical (+ve up).

Comment on Production

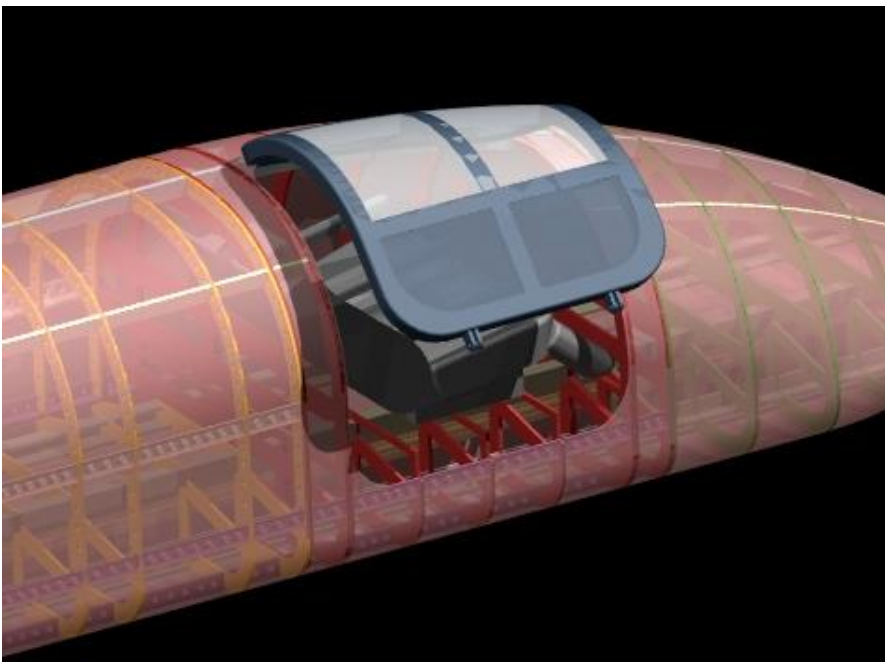
Since most of our structure is made of steel sheets of different thicknesses, it is easy to use laser cutting to cut the required shapes of the different frames from steel strips. Flanges are standard parts, however, some holes are made to be compatible with other parts of the structure. Also most of the longerons are standard section channels. All the parts are assembled using bolts or rivets. The skin shells are riveted to the flanges which is even integrated in the bulkheads or fastened to the frames. Stiffeners and other reinforcement parts are standard.

Maintenance

Since the structure is assembled using fasteners, no welding is needed. If a part is cracked or damaged, it can be easily replaced. However, if welding is a must, Steel is a good material for welding, due to its good thermal properties.

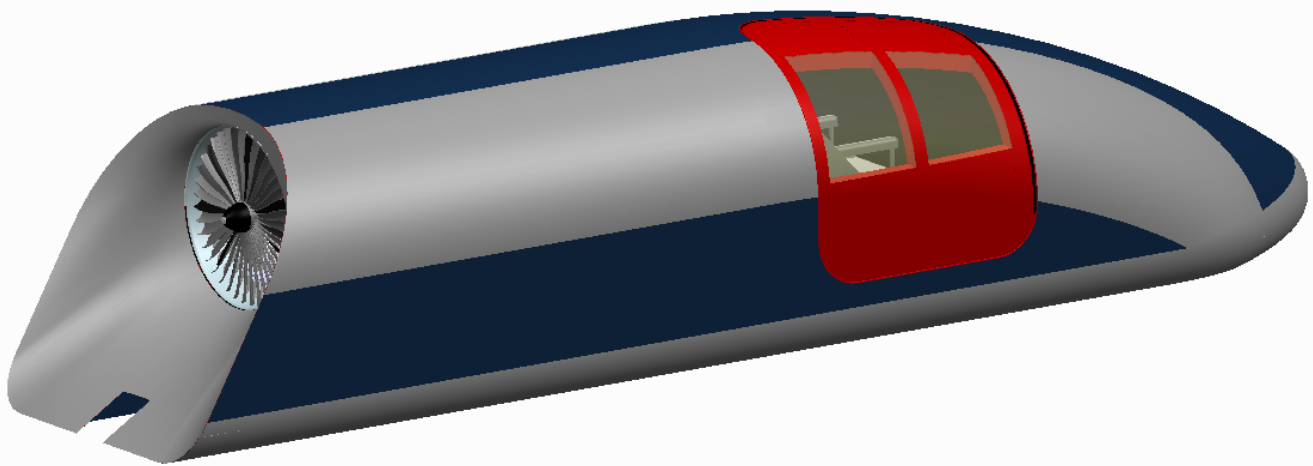
Cost

The manufacturing is mainly laser cutting of steel sheets, so that it does not cost much if compared to other subsystems. The cost of steel sheets can be estimated by 4200\$.



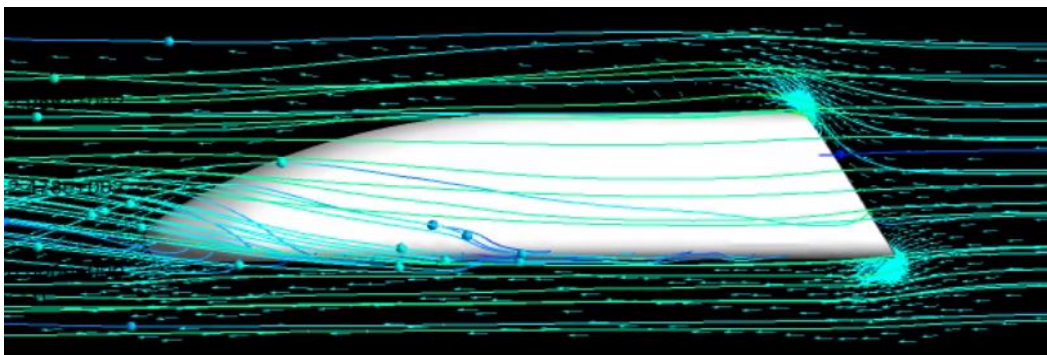
Aerodynamic Design

Our challenge in the aerodynamic design was to minimize the drag force coefficient of the pod. We used **Computational Fluid Dynamics (CFD)** to simulate the flow around the pod in the tube. We kept in mind that having a shock wave anywhere on the surface of the pod will result in a very high force of drag due to pressure difference, the capsule will then act like a syringe. We also tried to retard the flow separation on the pod as much as possible by making the pod streamlined. The following figure shows the final aerodynamic shape of the pod after many iterations.

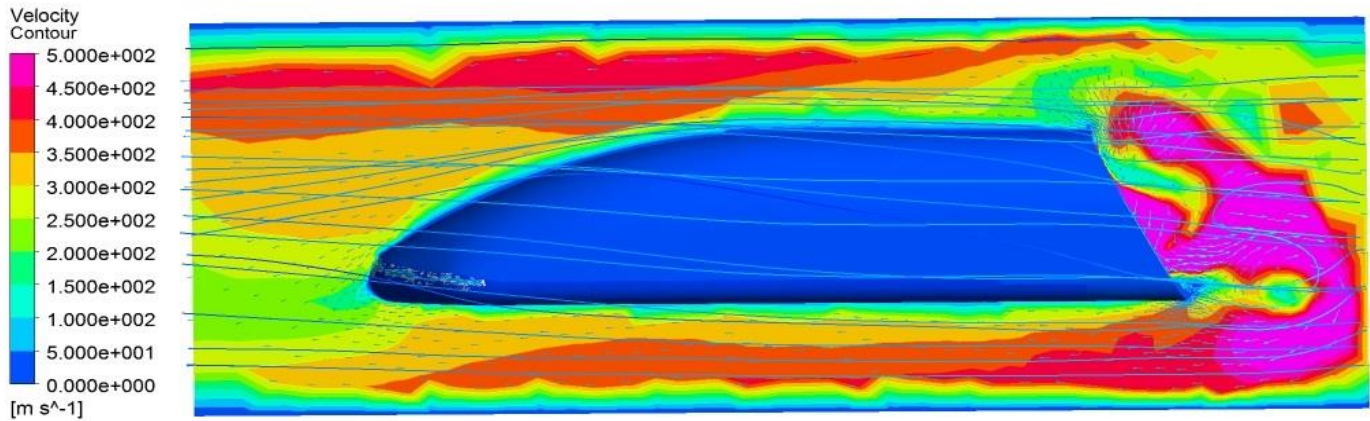


Moreover, We studied the flow turbulence around the pod and we tried to minimize the formation of vortices and the formation of a reversed flow.

A compressor is placed at the nose of the pod to compress the high pressure air and transfer from the front to the bottom nozzles for levitation by the air bearing system. Choosing the sufficient inlet air pressure for the compressor to be 500 Pa. We could determine the inlet conditions, hence we calculated the nozzles outlet conditions at the bottom from the levitation calculations.



One of the problems that faced our team, was the overall simulation of the flow around the pod in the presence of the compressor and the nozzles. We considered the compressor as a mass inlet and the nozzles as a pressure outlet. We calculated the mass flow rate required to be entering the compressor, and we calculated the cushion pressure at the bottom of the pod. Hence, we could simulate the flow around the pod. We got the aerodynamic coefficients and the velocity, pressure and temperature distributions.



Results

Tube pressure = 500 Pa

Compressor suction mass flow rate = 150 gm/s

Free stream velocity = 285 m/s

Mach number = 0.850

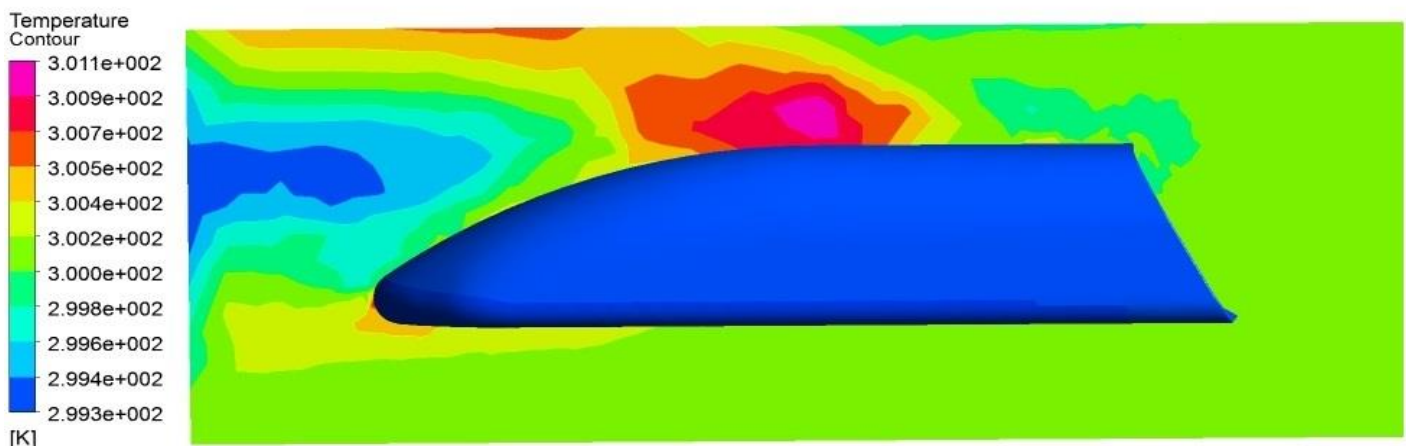
Reference area = 1.046 m²

Lift coefficient = -0.940 (without considering the air bearing very high cushion pressure)

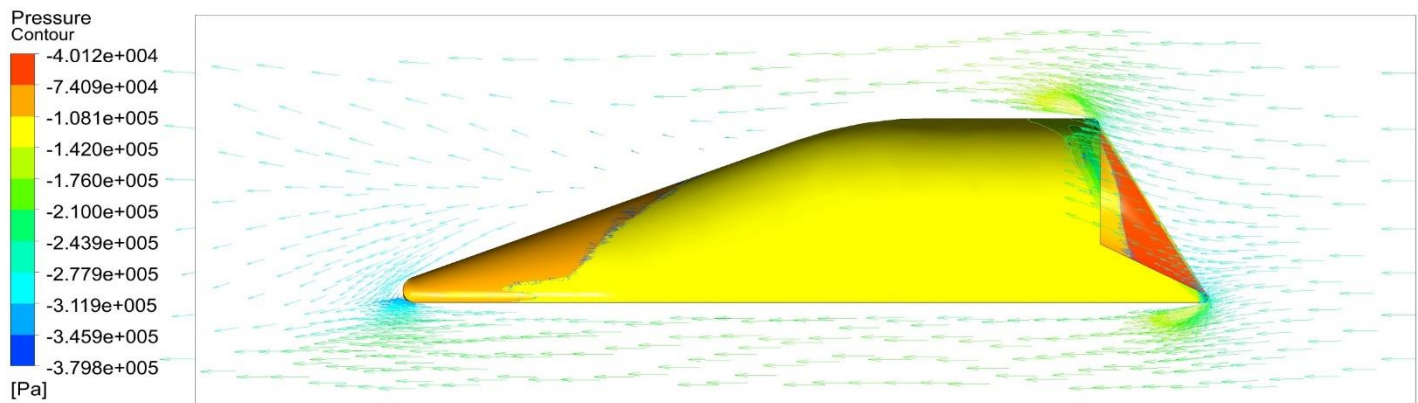
Drag coefficient = 0.864

Drag force = 22 kgf

Pitching moment coefficient = -0.170



We chose Aluminum as the skin material for its surface roughness. By simulation, it showed a good effect in reducing viscosity and turbulence effects.



The pod was simulated for different scenarios. The effect of the airbearing in increasing the drag force and the temperature at the bottom, was taken into consideration by simulating specific parts of the pod each alone and by entering correction factors to our calculations.

We simulated the pod in the worst cases, so that we can guarantee that no supersonic flow takes place anywhere on the surface of the pod. We studied the effect of the door and the bottom cushion skirt on the pod in the tube.

Traction system to endure air resistance while maintaining balanced speed

LIM propulsion system has many advantages over conventional propulsion systems, has excellent acceleration and deceleration as well as the ability to climb steep gradients. It has lower construction cost than conventional systems due to its small tunnel cross-section. It is quiet and runs smoothly (no mechanical couplings).

*After analyzing the traction force required for LIM from the running resistance (**R**) compared to **R** with no LIM, Comparing traction forces for various relative speeds:*

$$R = \left(1.3 \sqrt{\frac{10}{m}} + 0.01V \right) W + K(CoV^2) \quad (kgf)$$
$$Co = 0.0035S + 0.0041 \frac{pl}{100} + 0.002Np$$

where R is the running resistance (kgf), W is the total weight of the railway vehicles (tons), m is the axle load (tons), V is the railway vehicle speed (km/h), co is the air resistance coefficient on open ground, k is the increasing coefficient value of the air resistance, S is the cross-section of a vehicle front surface (m²), p is the circumference length of a vehicle (m), L is the length of the railway vehicles (m), and NP is the number of pantographs

$$P_{motor} = Requi \left(\frac{V}{3.6} \right) * efficiency \quad [W]$$
$$Ts = 28.35 * W * 1.05 * \alpha + Rs * W \quad (kgf)$$

Through analyzing the running resistance properties of a high-speed traction system then analyzing the power requirement for the traction electric motor to maintain a balanced speed in the high-speed traction system. The element techniques, for the development of a high-speed traction system for the Hyperloop, including the design criteria. We still have to iterate the required motor power to induce LIM thrust force required for the traction forces on the human scaled pod.

Safety Strategy

We have considered the Federal Railroad Administration (FRA) with the following principles

- 1. Establishing safety standards and program guidance for HSR,*
- 2. Applying a system safety approach to address safety concerns on specific rail lines, and*
- 3. Ensuring that railroads involved in passenger train operations can effectively and efficiently manage train emergencies.*

Strategy

- The presence or absence of freight traffic, the volume of freight traffic, and the nature of freight operations (e.g., overhead vs. local switching).*
- The degree to which passenger equipment used on the corridor(s) is of similar construction.*
- The degree of isolation of the passenger system from other hazards (e.g., incursions of motor vehicles due to proximity of roads and bridges, the degree of security that can be established on the right-of-way, and the presence of natural hazards such as seismic events or high water).*

Prevention

1- Vehicle- Track interaction

The standards for higher speeds focus on vehicle-track interaction as well as track geometry, rail integrity, and appropriate inspection practices.

Identifying track geometry irregularities associated with unsafe wheel forces and acceleration, thorough reviews of vehicle qualification and revenue service test data, and consideration of international practices.

We have to assure:

- Qualification requirements for high-speed or high cant deficiency operations, or both.*
- Acceleration and wheel force safety limits.*
- Inspection, monitoring, and maintenance requirements.*
- Track geometry limits for high-speed operations.*

2- POD control

The performance thresholds are intended to increase safety performance targets as the maximum speed limits increase to compensate for increased risks, including the potential frequency and adverse consequences of a collision or derailment.

- *Eliminate all redundant crossings*
- *Insulation and Isolation achieved for each subsystem on POD, including passenger area totally isolated from propulsion and levitation systems hazards*
- *A route map, location and status indicators are shown for on POD passengers to assure their personal safety*
- *Install the most sophisticated traffic control/warning devices compatible with the location (e.g., median barriers, special signage (possible active advanced warning))*
- *Protect rail movement with full width barriers capable of absorbing the impact of PODs*
- *provides a mechanism for each POD to establish communication with the ground back station*
- *A forced braking mechanism is established separately in case of NO or Negative response is received*
- *No running PODs with track maintenance and hazards along*
- *Real Time system monitoring, with on board sensors, wayside detection devices and autonomous track geometry systems*
- *Basic end strength and crash energy management performance in longitudinal encounters*
- *Other structural characteristics (e.g., side strength, roof strength as a function of vehicle weight)*
- *Acceptable occupant accelerations and restraint strategies based on expected POD performance*

3- Emergency Management and System Safety

- *Hazard Management “from tube, POD, or outside hazard”*
- *Including emergency egress and rescue access*
- *Lighting, signage, deceleration and braking control*

Levitation system

According to the Pod weight the minimum cushion pressure was determined.

Initially the number of holes was assumed and a number of iterations were done and the following results were satisfying.

Number of cones	76
Clearance height	0.0254 m
Clearance height of peripheral skirt	0.0127 m
The cushion perimeter	0.62832 m
Perimeter of peripheral skirt	36.7993 m
Cushion area	1.1938 m ²
Cushion area between peripheral skirt and the cones	0.61913 m ²
Cushion pressure	8491.5323 N/m ²
augmentation factor Ka	3.5487
Volume flow rate from holes	0.098827 m ³ /sec.
Area of holes	0.001639 m ²

MATLAB code:

```
clc
close all

%inlet conditions
disp('inlet condtions are:')
P1 = 500; disp(['P1 = ', num2str(P1), ' N/m^2'])
T1 = 288; disp(['T1 = ', num2str(T1), ' Kelvine'])
R_air = 287;
Rho1 = P1/(R_air*T1); disp(['Rho1 = ', num2str(Rho1), ' Kg/m^3'])
Rho3 = 0.21409; disp(['Rho3 = ', num2str(Rho3), ' kg/m^3'])
disp('-----')
%inputs
M = 1500; %kg
W = M*9.81; %N
nc = 38*2; %number of cones
hc1 = 2.54*10^-2; %m
hc2 = hc1/2; %m
Lc1 = 62.832*10^-2; %m
Lc2 = 2*(786.904+46+888.148+58.388+60.527)*10^-2; %m
Dc1 = 0.6;
Dc2 = 0.6;
```



```

Ac1 = nc*157.080*10^-4; %m^2
Ac2 = 6191.328*10^-4; %m^2

disp(['Mass = ', num2str(M), ' kg'])
disp(['Weight = ' num2str(w), ' N'])
disp(['Number of cones = ' num2str(nc)])
disp(['Clearance height = ' num2str(hc1), ' m'])
disp(['Clearance height of peripheral skirt = ' num2str(hc2), ' m'])
disp(['The cushion perimeter = ' num2str(Lc1), ' m'])
disp(['Perimeter of peripheral skirt = ' num2str(Lc2), ' m'])
disp(['Cushion area = ' num2str(Ac1), ' m^2'])
disp(['Cushion area between peripheral skirt and the cones = ' num2str(Ac2), ' m^2'])
disp('-----')

Kp = nc^2*hc1^2*Lc1^2*Dc1^2/(nc^2*hc1^2*Lc1^2*Dc1^2+hc2^2*Lc2^2*Dc2^2);
Pcu = w/(Ac1+Kp*Ac2); disp(['Cushion pressure = ' num2str(Pcu), ' N/m^2'])
Pa = hc2*Lc2*Dc2*(w/(Ac1+Kp*Ac2))^1.5*(2*Kp/Rho3)^0.5; Pa = Pa*0.001341; disp(['power required to sustain the
air cushion = ' num2str(Pa), ' HP'])
Ka = (Ac1+Kp*Ac2)/(2*Kp*hc2*Lc2*Dc2); disp(['augmentation factor Ka = ' num2str(Ka)])
Q1 = Pa/Pcu; disp(['Volume flow rate from holes = ' num2str(Q1), ' m^3/sec.'])
Vc = sqrt(2*Pcu/Rho3); disp(['Cushion velocity = ' num2str(Vc), ' m/s'])
Ah = Q1/(Rho3*Vc); disp(['Area of holes = ' num2str(Ah), ' m^2'])
Vc2 = (2*Kp*Pcu/Rho3)^0.5; disp(['velocity of the air escaping under peripheral skirt = ' num2str(Vc2), ' m/s'])
disp('-----')
losses = 20; disp(['assume losses from compressor to cushion holes = ', num2str(losses), ' %']) %percent
Pcomp = Pcu/(1-losses/100); disp(['pressure required from compressor = ', num2str(Pcomp), ' N/m^2'])
PR = Pcomp/P1; disp(['then pressure ratio required = ', num2str(PR)])
clear all

```

```

inlet conditons are:
P1 = 500 N/m^2
T1 = 288 Kelvine
Rho1 = 0.0060492 kg/m^3
Rho3 = 0.21409 kg/m^3
-----

Mass = 1500kg
Weight = 14715 N
Number of cones = 76
Clearance height = 0.0254 m
Clearance height of peripheral skirt = 0.0127 m
The cushion perimeter = 0.62832 m
Perimeter of peripheral skirt = 36.7993 m
Cushion area = 1.1938 m^2
Cushion area between peripheral skirt and the cones = 0.61913 m^2
-----

Cushion pressure = 8491.5323 N/m^2
power required to sustain the air cushion = 839.1901 HP
augmentation factor Ka = 3.5487
Volume flow rate from holes = 0.098827 m^3/sec.
Cushion velocity = 281.6501 m/s
Area of holes = 0.001639 m^2
Velocity of the air escaping under peripheral skirt = 262.8152 m/s
-----

assume losses from compressor to cushion holes = 20 %
pressure required from compressor = 10614.4153 N/m^2
then pressure ratio required = 21.2288

```


Free Vortex law Axial Flow Compressor design

The only purpose of the compressor is to provide the pressure required for levitation of the Hyperloop pod. This was achieved through the following procedure.

Design procedure:

From the Pressure ratio required from the compressor and the inlet conditions and the tunnel dimensions, the primary dimensions of the compressor was determined and the mathematical calculations could be initiated as follows.

Main specifications:

Tip diameter	60 cm
Hub diameter	50 cm
Inlet pressure	500 pascal
Inlet temperature	288 k
Inlet density	0.0060492 kg/m ³
Overall pressure ratio	21.5
Number of stages	3

Next calculations at the tip were obtained and the main parameters could be determined. From the results at the tip, and since the compressor was designed using Free Vortex Design Law, the rest of the blade was constructed.

At this point the mass flow rate was determined and the blades' rotational speed was determined.

Then the performance of the compressor was tested away from the design point by varying the free stream velocity.

The following MATLAB code illustrates the design results.

MATLAB Code:

```
clc
clear all
close all

disp('                                Free vortex law design                                ')
disp('                                Constant hub diameter design                                ')

%Inlet specifications
Dt = 0.6; disp(['Tip diameter = ', num2str(Dt), ' m'])
Rt = Dt/2;
Dh = 0.5; disp(['Hub diameter = ', num2str(Dh), ' m'])
Rh = Dh/2;
Dm = (Dt+Dh)/2; disp(['Mean diameter = ', num2str(Dm), ' m'])
Rm = Dm/2;
P1 = 500; disp(['Inlet flow pressure = ', num2str(P1), ' N/m^2'])
T1 = 288; disp(['Inlet flow temperature = ', num2str(T1), ' Kelvine'])
R = 287; disp(['Specific gas constant = ', num2str(R), ' J/kg.k'])
Rho1 = P1/(R*T1); disp(['Inlet flow density = ', num2str(Rho1), ' kg/m^3'])
Etas = 0.86; disp(['Stage isentropic efficiency = ', num2str(Etas)])
Etac = 0.86; disp(['Compressor overall isentropic efficiency = ', num2str(Etac)])
lambda = 0.95; disp(['Work done factor = ', num2str(lambda)])
Etam = 0.95; disp(['Mechanical efficiency = ', num2str(Etam)])
PR = 21.2; disp(['Overall required pressure ratio = ', num2str(PR)])
gamma = 1.4; %assuming gamma is constant
cp = gamma*R/(gamma-1);
n = 3; disp(['Number of stages = ', num2str(n)])
AVR = 1; disp(['Axial velocity ratio = ', num2str(AVR)])
disp('-----')

disp('At the Tip:')
disp('-----')

%Calculations
C1 = 288; %m/s
Ca = C1;
Rxt = 0.8; disp(['Reaction factor = ', num2str(Rxt)])
phit = 0.65; disp(['Flow coefficient ', num2str(phit)])
psit = 1; disp(['Stage loading coefficient = ', num2str(psit)])
DFt = 0.8; disp(['Diffusion factor = ', num2str(DFt)])

alpha1t = 0; disp(['alpha1 = ', num2str(alpha1t), ' degree'])
beta1t = atan2d(Rxt/phit+psit/(2*lambda*phit),1); disp(['beta1 tip = ', num2str(beta1t), ' degree'])
beta2t = atan2d(Rxt/phit-psit/(2*lambda*phit),1); disp(['beta2 tip = ', num2str(beta2t), ' degree'])
thetat = beta1t-beta2t; disp(['flow turning angle = ', num2str(thetat), ' degree'])
dCw = Ca*(tand(beta1t)-tand(beta2t));
v1 = Ca/cosd(beta1t); disp(['relative velocity at inlet = ', num2str(v1), ' m/s']) %from velocity diagram
M1 = v1/sqrt(gamma*R*T1); disp(['Inlet Mach number = ', num2str(M1)])
```

```

v2 = Ca/cosd(beta2t); disp(['relative velocity at exit = ', num2str(v2), ' m/s']) %from velocity diagram
To1 = T1+v1^2/(2*cp);
dTo = To1*((PR^((gamma-1)/gamma)-1)/Etac); disp(['total temperature difference across compressor = ', num2str(dTo), '
kelvine'])
dTos = dTo/n;
To2 = dTos+To1;
T2 = To2-v2^2/(2*cp);
PRs = (Etac*dTos/To1+1)^(gamma/(gamma-1)); disp(['Pressure ratio across stage = ', num2str(PRs)])
w = dTos*cp; disp(['work done per unit mass flow = ', num2str(w), ' watt'])
ut = w/(lambda*dCw); disp(['U tip = ', num2str(ut), ' m/s'])
rpm = ut/(pi*Dt)*60; disp(['Rpm = ', num2str(rpm)])
mdot = pi*Rt^2*Rho1*Ca; disp(['mass flow = ', num2str(mdot), ' kg/s'])
p = mdot*w/Etam; p = p/750; disp(['power required = ', num2str(p), ' hp'])
dH = v2/v1; disp(['de Haller number = ', num2str(dH)])
C1 = Ca/cosd(alpha1t); disp(['absolute velocity at inlet = ', num2str(C1), ' m/s'])
vw1 = Ca*tand(beta1t); disp(['relative whirl velocity at inlet = ', num2str(vw1), ' m/s'])
vw2 = Ca*tand(beta2t); disp(['relative whirl velocity at exit = ', num2str(vw2), ' m/s'])
alpha2t = atan2d(ut-vw2,Ca); disp(['alpha2 = ', num2str(alpha2t), ' degree'])
C2 = Ca/cosd(alpha2t); disp(['absolute velocity at exit = ', num2str(C2), ' m/s'])
cw1t = C1*sind(alpha1t); disp(['absolute whirl velocity at inlet = ', num2str(cw1t), ' m/s'])
cw2t = C2*sind(alpha2t); disp(['absolute whirl velocity at exit = ', num2str(cw2t), ' m/s'])
sigma = abs(vw1-vw2)/(2*v1)*(DFt-1+v2/v1)^(-1); disp(['solidity = ', num2str(sigma)])
pcr = 1/sigma; disp(['pitch chord ratio = ', num2str(pcr)])

disp('-----')
disp('At the Hub: ')
disp('-----')
cw1h = cw1t*Rt/Rh; disp(['absolute whirl velocity at inlet = ', num2str(cw1h)])
cw2h = cw2t*Rt/Rh; disp(['absolute whirl velocity at inlet = ', num2str(cw2h)])
uh = 2*pi*Rh*rpm/60; disp(['U hub = ', num2str(uh)])
alpha1h = atan2d(cw1h,Ca); disp(['alpha1 = ', num2str(alpha1h)])
alpha2h = atan2d(cw2h,Ca); disp(['alpha2 = ', num2str(alpha2h)])
beta1h = atan2d(uh-cw1h,Ca); disp(['beta1 hub = ', num2str(beta1h)])
beta2h = atan2d(uh-cw2h,Ca); disp(['beta2 hub = ', num2str(beta2h)])
thetah = beta1h-beta2h; disp(['flow turning angle = ', num2str(thetah), ' degree'])
Rx = Ca/(2*uh)*(tand(beta1h)+tand(beta2h)); disp(['Reaction factor at the hub = ', num2str(Rx)])
C1 = Ca/cosd(alpha1h); disp(['absolute velocity at hub inlet = ', num2str(C1), ' m/s'])
C2 = Ca/cosd(alpha2h); disp(['absolute velocity at hub exit = ', num2str(C2), ' m/s'])
v1 = Ca/cosd(beta1h); disp(['relative velocity at the inlet = ', num2str(v1), ' m/s'])
v2 = Ca/cosd(beta2h); disp(['relative velocity at the exit = ', num2str(v2), ' m/s'])
vw1 = v1*sind(beta1h);
vw2 = v2*sind(beta2h);
dH = v2/v1; disp(['de Haller number = ', num2str(dH)])
w = lambda*uh*Ca*(tand(beta1h)-tand(beta2h));
psi = w/uh^2; disp(['Stage loading factor = ', num2str(psi)])
phi = Ca/uh; disp(['Flow coefficient = ', num2str(phi)])
DF = 1-v2/v1+abs(vw1-vw2)/(2*sigma*v1); disp(['Diffusion factor = ', num2str(DF)])

disp('-----')

```

```

disp('At the mean diameter: ')
disp('-----')
cw1m = cw1t*Rt/Rm; disp(['absolute whirl velocity at inlet = ', num2str(cw1m)])
cw2m = cw2t*Rt/Rm; disp(['absolute whirl velocity at inlet = ', num2str(cw2m)])
um = 2*pi*Rm*rpm/60; disp(['U mean = ', num2str(um)])
alpha1m = atan2d(cw1m,Ca); disp(['alpha1 = ', num2str(alpha1m)])
alpha2m = atan2d(cw2m,Ca); disp(['alpha2 = ', num2str(alpha2m)])
beta1m = atan2d(um-cw1m,Ca); disp(['beta1 mean = ', num2str(beta1m)])
beta2m = atan2d(um-cw2m,Ca); disp(['beta2 mean = ', num2str(beta2m)])
thetam = beta1m-beta2m; disp(['flow turning angle = ', num2str(thetam), ' degree'])
Rx = Ca/(2*um)*(tand(beta1m)+tand(beta2m)); disp(['Reaction factor at the mean diameter = ', num2str(Rx)])
C1 = Ca/cosd(alpha1m); disp(['absolute velocity at mean diameter inlet = ', num2str(C1), ' m/s'])
C2 = Ca/cosd(alpha2m); disp(['absolute velocity at diameter exit = ', num2str(C2), ' m/s'])
v1 = Ca/cosd(beta1m); disp(['relative velocity at the inlet = ', num2str(v1), ' m/s'])
v2 = Ca/cosd(beta2m); disp(['relative velocity at the exit = ', num2str(v2), ' m/s'])
vw1 = v1*sind(beta1m);
vw2 = v2*sind(beta2m);
dH = v2/v1; disp(['de Haller number = ', num2str(dH)])
w = lambda*um*Ca*(tand(beta1m)-tand(beta2m));
psi = w/um^2; disp(['Stage loading factor = ', num2str(psi)])
phi = Ca/um; disp(['Flow coefficient = ', num2str(phi)])
DF = 1-v2/v1+abs(vw1-vw2)/(2*sigma*v1); disp(['Diffusion factor = ', num2str(DF)])
disp('-----')

```

%Off design performance at the tip

```

k = 100;
C1 = linspace(0,288,k);
Ca = C1;

```

for i = 1:k

```

dCw = Ca(i)*(tand(beta1t)-tand(beta2t));
v1 = Ca(i)/cosd(beta1t);
M1 = v1/sqrt(gamma*R*T1); M1v(i) = M1;
v2 = Ca(i)/cosd(beta2t);
To1 = T1+v1^2/(2*cp); dTo1v(i) = To1;
dTo = To1*((PR^((gamma-1)/gamma)-1)/Etac); dTov(i) = dTo;
dTos = dTo/n; dTosv(i) = dTos;
To2 = dTos+To1; dTo2v(i) = To2;
T2 = To2-v2^2/(2*cp);
PRs = (Etac*dTos/To1+1)^(gamma/(gamma-1)); PRsv(i) = PRs;
w = dTos*cp;
mdot = pi*Rt^2*Rho1*Ca(i); Mdot(i) = mdot;
p = mdot*w/Etam; p = p/750; P(i) = p;
dH = v2/v1; DH(i) = dH;
C1 = Ca(i)/cosd(alpha1t);
vw1 = Ca(i)*tand(beta1t);
vw2 = Ca(i)*tand(beta2t);
alpha2t = atan2d(ut-vw2,Ca(i));

```

```
C2 = Ca(i)/cosd(alpha2t);  
cw1t = Ca(i)*sind(alpha1t);  
cw2t = C2*sind(alpha2t);
```

```
end
```

```
figure(1)  
plot(Ca,M1v,'c','linewidth',2)  
title('Mach number with inlet velocity at the tip')  
xlabel('Inlet velocity')  
ylabel('Mach number')  
grid
```

```
figure(2)  
plot(Ca,dTov,'k','linewidth',2)  
title('Temperature difference across compressor with inlet velocity at the tip')  
xlabel('Inlet velocity')  
ylabel('Temperature difference')  
grid
```

```
figure(3)  
plot(Ca,dTosv,'r','linewidth',2)  
title('Inlet velocity with difference in pressure across stage at the tip')  
xlabel('Inlet velocity')  
ylabel('Temperature difference across stage at the tip')  
grid
```

```
figure(4)  
plot(Ca,P,'m','linewidth',2)  
title('Inlet velocity with power required')  
xlabel('Inlet velocity')  
ylabel('Power required')  
grid
```

```
figure(5)  
plot(Ca,DH,'linewidth',2)  
title('Inlet velocity with de Haller number at the tip')  
xlabel('Inlet velocity')  
ylabel('de Haller number')  
grid
```

```
%off design performance at the hub
```

```
k = 100;  
C1 = linspace(0,288,k);  
Ca = C1;
```

```
for j=1:k
```

```
    Rx = Ca(j)/(2*uh)*(tand(beta1h)+tand(beta2h)); Rxv(i) = Rx;
```

```

C1 = Ca(j)/cosd(alpha1h);
C2 = Ca(j)/cosd(alpha2h);
v1 = Ca(j)/cosd(beta1h);
v2 = Ca(j)/cosd(beta2h);
vw1 = v1*sind(beta1h);
vw2 = v2*sind(beta2h);
dH = v2/v1; DH(i) = dH;
w = lambda*uh*Ca(j)*(tand(beta1h)-tand(beta2h));
psi = w/uh^2; Psi(i) = psi;
phi = Ca(j)/uh; Phi(i) = phi;
DF = 1-v2/v1+abs(vw1-vw2)/(2*sigma*v1); DFv(i) = DF;
end

```

```

figure(6)
plot(Ca,Rxv,'linewidth',2)
title('Inlet velocity with reaction factor at the hub')
xlabel('Inlet velocity')
ylabel('Reaction factor')
grid

```

```

figure(7)
plot(Ca,DH,'linewidth',2)
title('Inlet velocity with de Haller number at the hub')
xlabel('Inlet velocity')
ylabel('de Haller number')
grid

```

```

figure(8)
plot(Ca,Psi,'linewidth',2)
title('Inlet velocity with Stage loading coefficient at the hub')
xlabel('Inlet velocity')
ylabel('Stage factor coefficient')
grid

```

```

figure(9)
plot(Ca,Phi,'linewidth',2)
title('Inlet velocity with Flow coefficient at the hub')
xlabel('Inlet velocity')
ylabel('Flow coefficient')
grid

```

```

figure(10)
plot(Ca,DFv,'linewidth',2)
title('Inlet velocity with Diffusion factor at the hub')
xlabel('Inlet velocity')
ylabel('Diffusion factor')
grid

```

Free vortex law design
Constant hub diameter design

Tip diameter = 0.6 m
Hub diameter = 0.5 m
Mean diameter = 0.55 m
Inlet flow pressure = 500 N/m²
Inlet flow temperature = 288 kelvine
Specific gas constant = 287 J/kg.k
Inlet flow density = 0.0060492 kg/m³
Stage isentropic efficiency = 0.86
Compressor overall isentropic efficiency = 0.86
Work done factor = 0.95
Mechanical efficiency = 0.95
Overall required pressure ratio = 21.2
Number of stages = 3
Axial velocity ratio = 1

At the Tip:

Reaction factor = 0.8
Flow coefficient 0.65
Stage loading coefficient = 1
Diffusion factor = 0.8
 $\alpha_1 = 0$ degree
 β_1 tip = 63.8915 degree
 β_2 tip = 22.8337 degree
flow turning angle = 41.0578 degree
relative velocity at inlet = 654.4373 m/s
Inlet Mach number = 1.9238
relative velocity at exit = 312.488 m/s
total temperature difference across compressor = 811.8362 kelvine
Pressure ratio across stage = 3.7998
work done per unit mass flow = 271829.8093 watt
 U tip = 613.5048 m/s
 $R_{pm} = 19528.4636$
mass flow = 0.49258 kg/s
power required = 187.9285 hp
de Haller number = 0.47749
absolute velocity at inlet = 288 m/s
relative whirl velocity at inlet = 587.6599 m/s
relative whirl velocity at exit = 121.2632 m/s
 $\alpha_2 = 59.669$ degree
absolute velocity at exit = 570.3033 m/s
absolute whirl velocity at inlet = 0 m/s
absolute whirl velocity at exit = 492.2416 m/s
solidity = 1.2841
pitch chord ratio = 0.77874

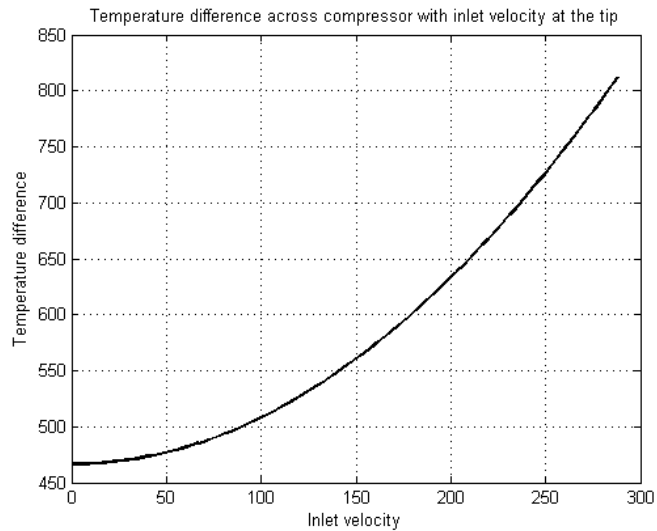
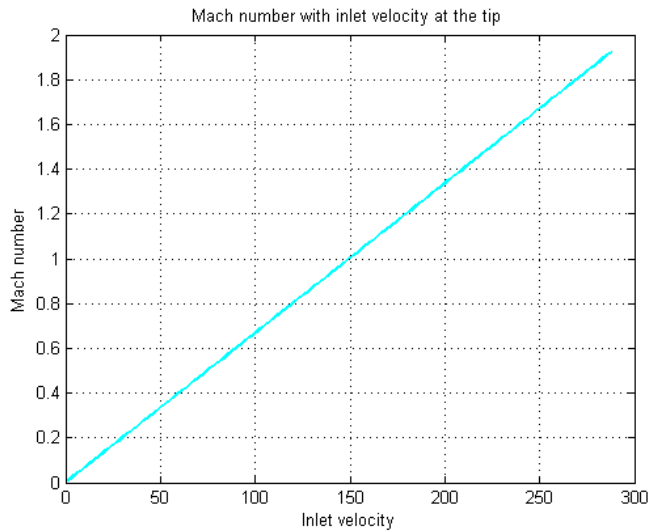
At the Hub:

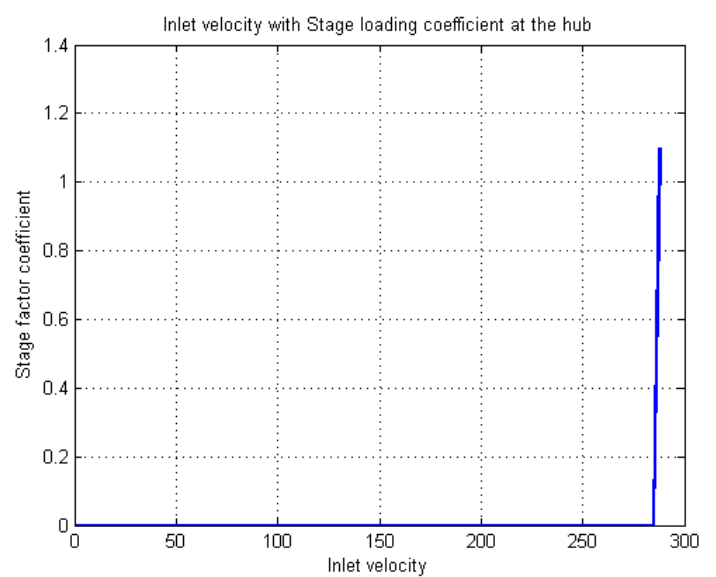
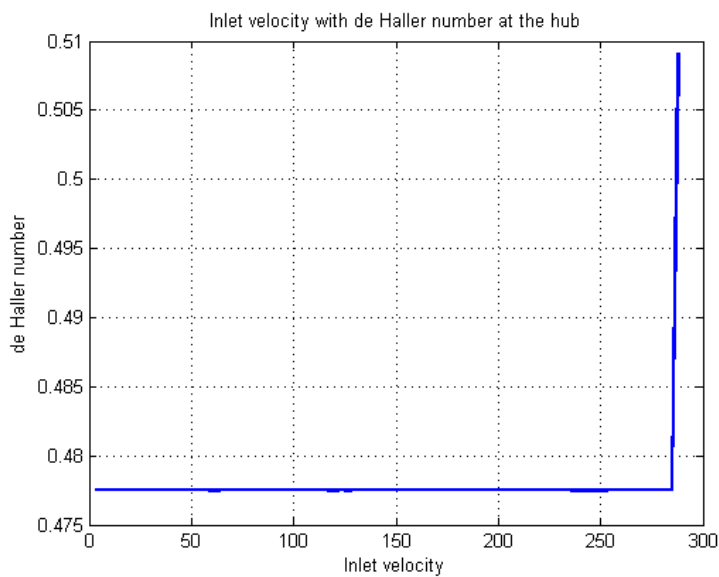
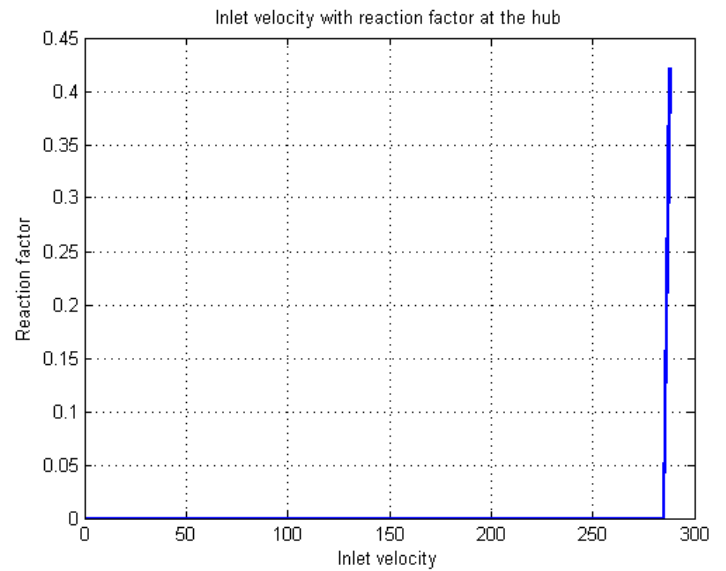
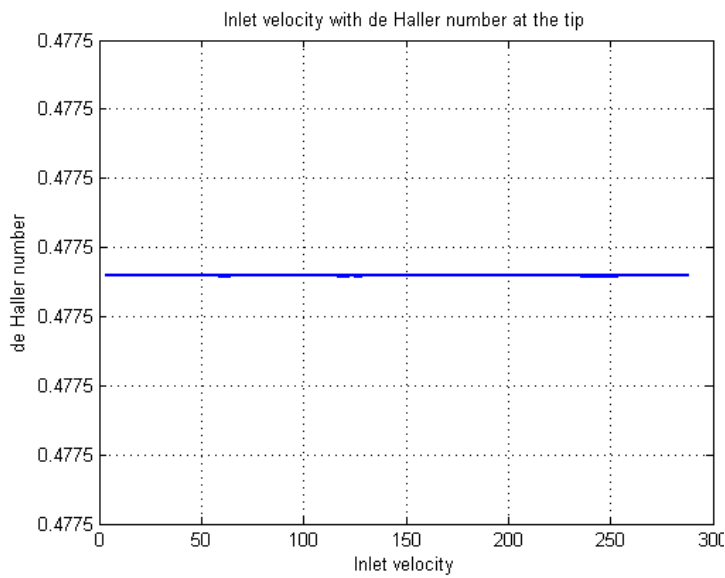
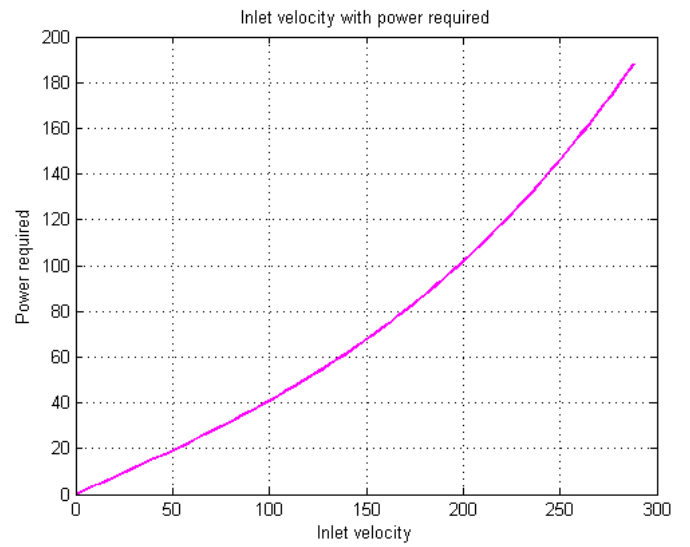
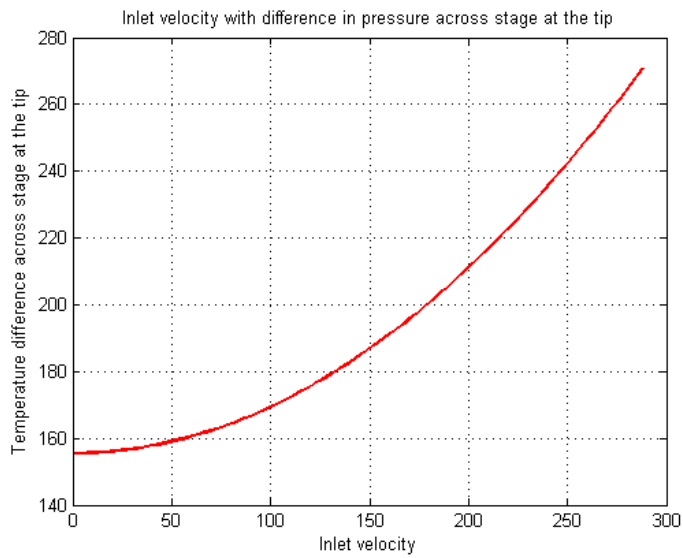
absolute whirl velocity at inlet = 0
absolute whirl velocity at inlet = 590.6899
 U hub = 511.254
 $\alpha_1 = 0$
 $\alpha_2 = 64.0077$
 β_1 hub = 60.6065
 β_2 hub = -15.4199
flow turning angle = 76.0264 degree
Reaction factor at the hub = 0.42231
absolute velocity at hub inlet = 288 m/s

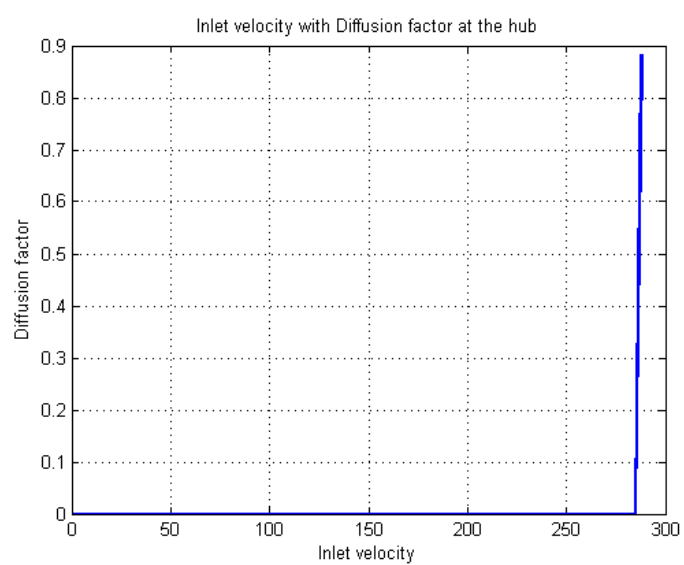
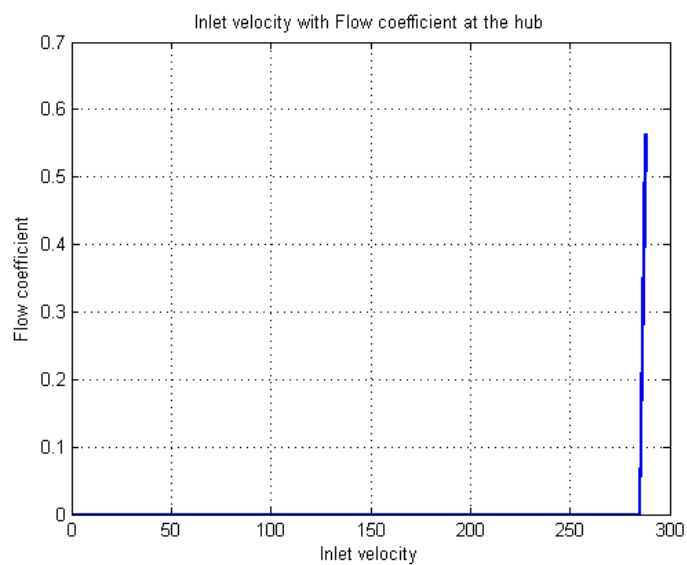
absolute velocity at hub exit = 657.1595 m/s
relative velocity at the inlet = 586.7918 m/s
relative velocity at the exit = 298.7542 m/s
de Haller number = 0.50913
Stage loading factor = 1.0976
Flow coefficient = 0.56332
Diffusion factor = 0.88282

At the mean diameter:

absolute whirl velocity at inlet = 0
absolute whirl velocity at inlet = 536.9909
U mean = 562.3794
alpha1 = 0
alpha2 = 61.7944
beta1 mean = 62.8826
beta2 mean = 5.0379
flow turning angle = 57.8447 degree
Reaction factor at the mean diameter = 0.52257
absolute velocity at mean diameter inlet = 288 m/s
absolute velocity at diameter exit = 609.3465 m/s
relative velocity at the inlet = 631.8343 m/s
relative velocity at the exit = 289.1169 m/s
de Haller number = 0.45758
Stage loading factor = 0.90711
Flow coefficient = 0.51211
Diffusion factor = 0.87334







Braking System

When the magnet is moved along the rail, it generates a non-stationary magnetic field in the head of the rail, which then generates electrical tension (Faraday's induction law), and causes eddy currents. These disturb the magnetic field in such a way that the magnetic force is diverted to the opposite of the direction of the movement, thus creating a horizontal force component, which works against the movement of the magnet

The eddy current brake does not have any mechanical contact with the rail, and thus no wear, and creates no noise or odor. The eddy current brake is unusable at low speeds, but can be used at high speeds both for emergency braking and for regular braking

But in order to avoid the risk posed by potential power outages, they utilize permanent magnets instead of electromagnets, thus not requiring any power supply, however, without the possibility to adjust the braking strength as easily as with electromagnets

Characteristics

- **Peak force:** is the maximum decelerating effect that can be obtained. The peak force is often greater than the traction limit of the tires, in which case the brake can cause a wheel skid.
- **Continuous power dissipation:** Brakes typically get hot in use, and fail when the temperature gets too high. The greatest amount of power (energy per unit time) that can be dissipated through the brake without failure is the continuous power dissipation. Continuous power dissipation often depends on e.g., the temperature and speed of ambient cooling air.
- **Fade:** As a brake heats, it may become less effective, called brake fade. Some designs are inherently prone to fade, while other designs are relatively immune. Further, use considerations, such as cooling, often have a big effect on fade.

Main Concept of Electromagnetic braking

The idea is to simply use the SLIM to reverse thrust linearly providing deceleration through reversing poles and adding frequency changer to decelerate linearly to stop permanently at the end of the one mile track.

The parts of Electromagnetic Disc Brake are:

- AC Motor
- Disc
- Frame
- Electromagnet
- Pulleys & Belt
- Shaft

Thermal stability of the electromagnetic brakes is achieved by means of the convection and radiation of the heat energy

at high temperature. The value of the energy dissipated by the fan can be calculated by the following expression

$$Q = MCp \Delta T$$

Where M = Mass of air circulated;

Cp = Calorific value of air;

ΔT = Difference in temperature between the air entering and the air leaving the fan

The electromagnet is energized by the AC supply where the magnetic field produced is used to provide the braking mechanism. When the electromagnet is not energized, the rotation of the disc is free and accelerates uniformly under the action of weight to which the shaft is connected. When the electromagnet is energized, magnetic field is produced thereby applying brake by retarding the rotation of the disc and the energy absorbed is appeared as heating of the disc. So when the armature is attracted to the field the stopping torque is transferred into the field housing and into the machine frame decelerating the load. The AC motor makes the disc to rotate through the shaft by means of pulleys connected to the shaft.

It was found that electromagnetic brakes can develop a negative power which represents nearly twice the maximum power output of a typical engine, and at least three times the braking power of an exhaust brake. These performances of electromagnetic brakes make them much more competitive candidate for alternative retardation equipment's compared with other retarders. The brake linings would last considerably longer before requiring maintenance, and the potentially "brake fade" problem could be avoided. In research conducted by a truck manufacturer, it was proved that the electromagnetic brake assumed 80 percentage of the duty which would otherwise have been demanded of the regular service

brake. Furthermore, the electromagnetic brake prevents the dangers that can arise from the prolonged use of brakes beyond their capability to dissipate heat. This is most likely to occur while a vehicle descending a long gradient at high speed.

Power Supply

For the SLIM:

Frequency(HZ)	500
Input Power (KW)	173
Output power (KW)	155
Line voltage (V)	960
No. of Batteries	6
Transformers attached	Available
Frequency Changer	Available
3-phase inverter	Available

References

1. *Journal of Electrical Engineering & Technology Vol. 7, No. 2, pp. 200~206, 2012*
<http://dx.doi.org/10.5370/JEET.2012.7.2.200>
2. C.B. Park, B.S. Lee, and J. Lee, "A study on the applicability of the conventional TTX propulsion system on the high-speed propulsion system for a deep-underground GTX." *International Journal of Railway*, Vol. 3, No. 2, pp.54-59, June 2010
3. U.S. Department of Transportation, Federal Railroad Administration, Version 1.0, November 2009
4. *International Journal of Innovative Research in Science, Engineering and Technology An ISO 3297: 2007 Certified Organization Volume 3, Special Issue 2, April 2014*
Second National Conference on Trends in Automotive Parts Systems and Applications (TAPSA-2014)
5. Budig P. K. The application of linear motors // *Proceedings of the International Conference on Power Electronics and Motion Control*, 2000. – Vol. 3. – P. 1336 – 1341.

Navigation and Control

Design of an Inertial Navigation System Using MEMS Sensors

1 Introduction

1.1 Inertial Navigation

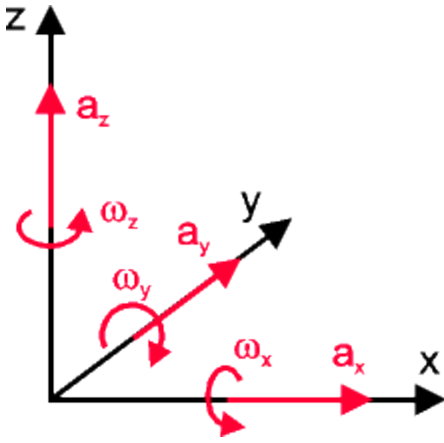
Inertial navigation systems are used in many situations where the use of an external reference to measure position is impractical or unreliable. Typical inertial navigation systems used in aeronautics and marine applications are highly advanced pieces of equipment costing thousands of dollars. However, inexpensive accelerometers and angular rate sensors (gyros) can be used to make a less accurate inertial navigation for lower cost.

An Inertial Navigation System (INS) is a device which computes the real-time state of a moving vehicle using motion sensors. By state we mean the position, velocity, and orientation of the vehicle. An INS can be used either as an isolated unit to provide continuous state information to a pilot, or in conjunction with a control system for autonomous movement. INSs are widely used in military and commercial projects, and have been under constant development and revision for several decades, but it is only recently that the technology has been accessible to hobbyists. An INS can be logically decomposed into an Inertial Measurement Unit (IMU), which measures instantaneous accelerations or velocities in the body frame, and a state update system which uses the IMU values to update the position, velocity, and orientation of the vehicle in the navigation frame.

An inertial navigation system (INS) uses two types of sensors called accelerometers and gyros to measure its motion parameters. A prototypical accelerometer contains a mass suspended on a spring, with some way of measuring the extent to which the spring is compressed. When the accelerometer's body is accelerated, a force is transmitted to the mass through the spring, causing the spring to stretch or contract. This can then be measured, and results in a value proportional to the accelerometer's acceleration. A gyro is a device that measures the rate of rotation around the gyro axis. The earliest gyros were actual spinning gyroscopes which, when rotated perpendicularly to their spin axis, will produce a force which can be measured.

1.2 Mathematical Background

A body's actual spatial behavior / movement can be described with six parameters: for example with three translatory (x -, y -, z -acceleration) and three rotatory components (x -, y -, z -angular velocity). To be able to define the movement of the body, three acceleration sensors and three gyros have to be put together on a platform in such a way, that they form an orthogonal system. By integrating the individual translatory and rotatory components the current position and orientation can be calculated. Mathematic integration of the acceleration data yields the current speed, a second mathematic integration provides ultimately the distance travelled from the starting point. Since the angular sensors used within this project provide output data representing rotation speed (not angular acceleration), a single mathematic integration already yields the angular orientation. Performing these calculations accurately and periodically enables the ideal system to trace its movement with respect to a (virtual) reference point and to indicate its speed, current position and heading.



$$s(t) = \iint a(t) dt^2$$

$$\varphi(t) = \int \omega(t) dt$$

The six degrees of freedom

A body's actual spatial behavior / movement can be described with three translatory and three rotary components.

The main limitation of the system performance is given with the finite precision of the sensors. A continuous small error in acceleration will be integrated once and results in a big error in actual speed, integrated a second time in a huge error in distance. Therefore very precise sensors and error correction mechanisms (feedback algorithms) are necessary to get an accurate inertial navigation platform.

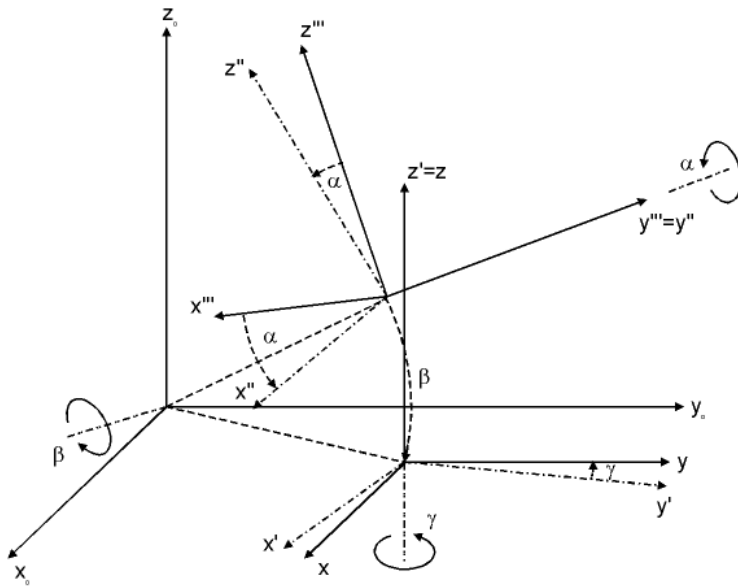
1.2.1 Coordinate System

There exist several valuable choices of coordinate systems. The choice of the right system coordinates is a very important task, which needs careful evaluation. There exist different solutions: Some of them introduce redundancy to allow for improved precision in order to mitigate rounding errors. Some coordinate systems also have singularities, for example at the conventional earth coordinates with "North-pole" and "South-pole".

We have chosen the representation with "Euler-angles", even though it has singularities. It is, however, the most intuitive coordinate system and did not limit our measurements. It defines 3 axes: roll/bank (alpha), pitch (beta) and direction (gamma). There are always two coordinate systems: the initial system (x, y, z), usually where the unit was initialized, and the coordinate system of the unit itself (x''', y''', z'''). The three linear accelerations and the three angular speeds first have to be transformed into the initial system. Then they are integrated.

The translation to the components referring to the initial system proceeds as follows:

$$(x''', y''', z''') = (\alpha) \Rightarrow (x'', y'', z'') = (\beta) \Rightarrow (x', y', z') = (\gamma) \Rightarrow (x, y, z).$$



Representation of the body's actual spatial situation (x''' , y''' , z''') referring to the initial system (x , y , z) by using 'Euler'-angles.

$$\begin{pmatrix} \Delta x \\ \Delta y \\ \Delta z \end{pmatrix} = \begin{pmatrix} \cos(\gamma) * \cos(\alpha) & -\sin(\gamma) * \cos(\beta) & -\cos(\alpha) * \sin(\beta) * \sin(\gamma) \\ -\sin(\gamma) * \sin(\beta) * \sin(\alpha) & \cos(\gamma) * \cos(\beta) & +\sin(\alpha) * \cos(\gamma) \\ \sin(\beta) * \sin(\alpha) * \cos(\gamma) & \cos(\gamma) * \cos(\beta) & \cos(\gamma) * \sin(\beta) * \cos(\alpha) \\ -\cos(\alpha) * \sin(\gamma) & & +\sin(\gamma) * \sin(\alpha) \\ \cos(\beta) * \sin(\alpha) & -\sin(\beta) & \cos(\beta) * \cos(\alpha) \end{pmatrix} \begin{pmatrix} \Delta x''' \\ \Delta y''' \\ \Delta z''' \end{pmatrix}$$

Calculation of the actual acceleration (Δx , Δy , Δz) referring to the initial system (x , y , z) and using the mathematic transformation based on 'Euler'-angles.

$$\begin{pmatrix} \Delta \alpha \\ \Delta \beta \\ \Delta \gamma \end{pmatrix} = \begin{pmatrix} 1 & \sin(\alpha) * \tan(\beta) & -\cos(\alpha) * \tan(\beta) \\ 0 & \cos(\alpha) & \sin(\alpha) \\ 0 & -\sin(\alpha) / \cos(\beta) & \cos(\alpha) / \cos(\beta) \end{pmatrix} \begin{pmatrix} \Delta \alpha''' \\ \Delta \beta''' \\ \Delta \gamma''' \end{pmatrix}$$

Calculation of the actual angular velocity ($\Delta \alpha$, $\Delta \beta$, $\Delta \gamma$) referring to the initial system (x , y , z) and using the mathematic transformation based on 'Euler'-angles.

1.3 MEMS Sensors

MEMS IMUs generally require three orthogonal measurements of acceleration and three measurements of rotational velocity around the same orthogonal axes. Accelerometers and gyroscopes respectively can produce these measurements. Both use micro-machined surface capacitors to measure forces, whether due to the acceleration or the Coriolis Effect.

2 Hardware Design

2.1 Components Selection

2.1.1 IMU

In order to design the INS hardware, we had to make choices about which components to use. The most important component of the design is, of course, the MEMS sensors themselves. We chose the **x-IMU** "<http://www.x-io.co.uk/products/x-imu/>", as it was designed to be the most versatile Inertial Measurement Unit (IMU) and Attitude Heading Reference System (AHRS) platform available. Its host of on-board sensors, algorithms, configurable auxiliary port and real-time communication via USB, Bluetooth or UART make it both a powerful sensor and controller. The on-board SD card, battery charger (via USB), real-time clock/calendar and motion trigger wake up also make the x-IMU an ideal standalone data logger.

The x-IMU GUI can be used to configure settings, view real-time sensor data, perform calibration and export data to user software; e.g. Microsoft Excel. The x-IMU MATLAB library provides all the tools required to import, organise and plot data in MATLAB. User software can be developed using the x-IMU API.

x-IMU Specifications:

- On-board sensors
 - Triple axis 16-bit gyroscope – Selectable range up to ± 2000 °/s
 - Triple axis 12-bit accelerometer – Selectable range up to ± 8 g
 - Triple axis 12-bit magnetometer – Selectable range up to ± 8.1 G
 - 12-bit battery voltage level
 - Factory calibrated
 - Temperature compensated (gyroscope only)
 - Selectable data rates up to 512 Hz
- On-board algorithms
 - IMU and AHRS algorithms provide real-time measurement of orientation relative to the Earth
 - Internal states updated at 512 Hz
 - Algorithm 'initialise' and 'tare' commands can be sent in real-time
 - Complete sensor calibration algorithms for user maintenance
- Auxiliary port
 - External power in from 3.6 V to 6.3 V
 - 3.3 V power out up to 100 mA
 - Digital I/O mode – 8 channels, controlled via USB or Bluetooth
 - Analogue input mode – 8 channels, 12-bit resolution, 0 to 3.3 V
 - PWM output mode – 4 channels, 1 to 65,535 Hz, controlled via USB or Bluetooth
 - UART mode – 3.3 V, 2400 to 230.4k baud, substitutes Bluetooth
- Power options
 - USB
 - LiPo battery – On-board charging via USB
 - External source from 3.6 V to 6.3 V
 - Low power consumption – 50 mA to 150 mA dependent on settings and usage, 130 μ A sleep mode

- Low profile
 - Dimensions: 33 × 42 × 10 mm (57 × 38 × 21 mm with plastic housing and battery)
 - Weight: 12g (49 g with plastic housing and battery)
- Other features
 - Motion triggered wake-up and sleep timer
 - Real-time clock and calendar
 - Configurable command button
 - Configurable 8 channel auxiliary port
- Connectivity
 - USB
 - Bluetooth – Class 1, 100m range, SPP
 - Micro SD card – Supports FAT16/32 and SDHC
 - UART (see auxiliary port mode)
- Software
 - View, edit and backup all internal x-IMU settings
 - Real-time 2D and 3D data graphics
 - Control panels for auxiliary port
 - Data logger and file converter for exporting data; e.g. to MATLAB, Microsoft Excel, etc.
 - Magnetic calibration tools
 - Firmware bootloader to access new features in future x-IMU firmware versions

2.1.2 Microcontroller

Our choice was based on the quality of the hardware and to support our project by running our software. We chose the RASPBERRY PI 2 MODEL B “<https://www.raspberrypi.org/products/raspberry-pi-2-model-b/>”. The Raspberry Pi 2 Model B is the second generation Raspberry Pi. It replaced the original Raspberry Pi 1 Model B+ in February 2015. Compared to the Raspberry Pi 1 it has:

- A 900MHz quad-core ARM Cortex-A7 CPU
- 1GB RAM

Like the (Pi 1) Model B+, it also has:

- 4 USB ports
- 40 GPIO pins
- Full HDMI port
- Ethernet port
- Combined 3.5mm audio jack and composite video
- Camera interface (CSI)
- Display interface (DSI)
- Micro SD card slot
- VideoCore IV 3D graphics core

Because it has an ARMv7 processor, it can run our MATLAB based project.

2.1.3 Color Sensor

We had a difficulty choosing a color detection sensor that can fit in our application of very high speeds as its reading time must be very small, so we chose the best we could find. We chose this Color Detector Sensor "<https://www.sparkfun.com/products/retired/11195>" as it has:

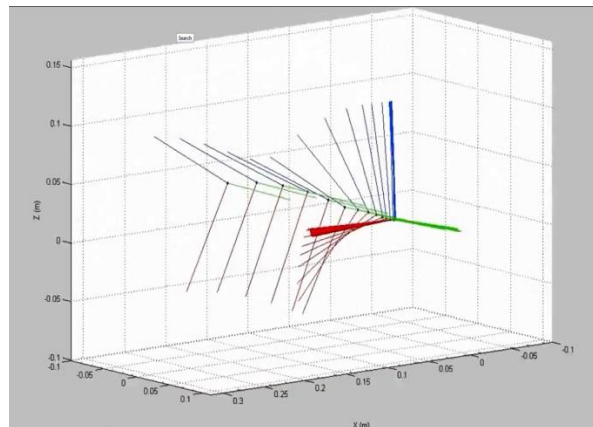
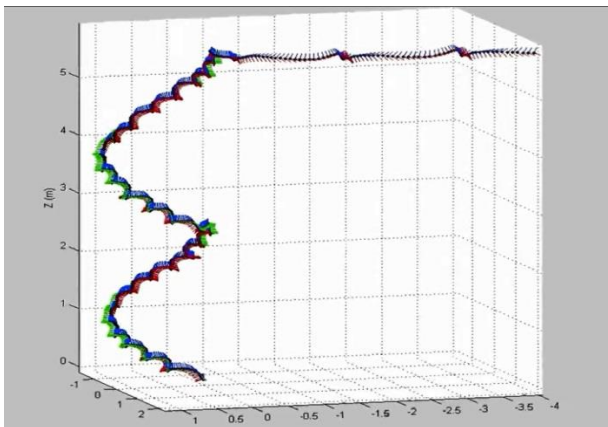
- R,G,B data is in true 8-bit RGB format from 0-255
- Wide operating temperature range: -40 Celsius to +85 Celsius
- Reading time: 620ms
- Wide operating voltage range: 3.1V to 5.5V
- Shock resistant to 16 mega Pascals

By mounting one on each side of the pod we will be able to detect and track color variations along the tube and calibrate the data from the x-IMU to determine the exact location of the pod inside the tube and to be able to control the velocity of the pod through each section of the tube safely by activating the braking system on time.

3 Software Design

Our software design is mainly based on the open source project **Gait tracking with x-IMU** "<http://www.x-io.co.uk/gait-tracking-with-x-imu/>" using its open source code as it is perfectly suitable for our application to track the pod inside the tube getting its position, attitude, velocity and acceleration.

The source code is a MATLAB code based on the x-IMU Matlab Library and some functions of Euler-angle approach, consisting of a developed function of a 6-D of Animation Function to provide a perfect GUI for this application.



The Matlab Code will be run by the RASPBERRY PI 2 MODEL B running and controlling the x-IMU. The RASPBERRY PI 2 MODEL B also will run and control the two color sensors mounted on each side of the pod, and by emerging and calibrating the data coming from the x-IMU and from the color sensor we will be able to determine the exact location of the pod inside the tube by the red and yellow lines in the tube.

3.1 Used Function

3.1.1

```
function q = axisAngle2quatern(axis, angle)
    q0 = cos(angle./2);
    q1 = -axis(:,1)*sin(angle./2);
    q2 = -axis(:,2)*sin(angle./2);
    q3 = -axis(:,3)*sin(angle./2);
    q = [q0 q1 q2 q3];
end
```

3.1.2

```
function R = axisAngle2rotMat(axis, angle)
    kx = axis(:,1);
    ky = axis(:,2);
    kz = axis(:,3);
    cT = cos(angle);
    sT = sin(angle);
    vT = 1 - cos(angle);

    R(1,1,:) = kx.*kx.*vT + cT;
    R(1,2,:) = kx.*ky.*vT - kz.*sT;
    R(1,3,:) = kx.*kz.*vT + ky.*sT;

    R(2,1,:) = kx.*ky.*vT + kz.*sT;
    R(2,2,:) = ky.*ky.*vT + cT;
    R(2,3,:) = ky.*kz.*vT - kx.*sT;

    R(3,1,:) = kx.*kz.*vT - ky.*sT;
    R(3,2,:) = ky.*kz.*vT + kx.*sT;
    R(3,3,:) = kz.*kz.*vT + cT;
end
```

3.1.3

```
function R = euler2rotMat(phi, theta, psi)
    R(1,1,:) = cos(psi).*cos(theta);
    R(1,2,:) = -sin(psi).*cos(phi) + cos(psi).*sin(theta).*sin(phi);
    R(1,3,:) = sin(psi).*sin(phi) + cos(psi).*sin(theta).*cos(phi);

    R(2,1,:) = sin(psi).*cos(theta);
    R(2,2,:) = cos(psi).*cos(phi) + sin(psi).*sin(theta).*sin(phi);
    R(2,3,:) = -cos(psi).*sin(phi) + sin(psi).*sin(theta).*cos(phi);

    R(3,1,:) = -sin(theta);
    R(3,2,:) = cos(theta).*sin(phi);
    R(3,3,:) = cos(theta).*cos(phi);
end
```

3.1.4

```
function euler = quatern2euler(q)
    R(1,1,:) = 2.*q(:,1).^2-1+2.*q(:,2).^2;
    R(2,1,:) = 2.*(q(:,2).*q(:,3)-q(:,1).*q(:,4));
    R(3,1,:) = 2.*(q(:,2).*q(:,4)+q(:,1).*q(:,3));
    R(3,2,:) = 2.*(q(:,3).*q(:,4)-q(:,1).*q(:,2));
    R(3,3,:) = 2.*q(:,1).^2-1+2.*q(:,4).^2;

    phi = atan2(R(3,2,:), R(3,3,:));
    theta = -atan(R(3,1,:) ./ sqrt(1-R(3,1,:).^2));
    psi = atan2(R(2,1,:), R(1,1,:));

    euler = [phi(1,:) theta(1,:) psi(1,:)]';
end
```

3.1.5

```
function R = quatern2rotMat(q)
    [rows cols] = size(q);
    R = zeros(3,3, rows);
    R(1,1,:) = 2.*q(:,1).^2-1+2.*q(:,2).^2;
    R(1,2,:) = 2.*(q(:,2).*q(:,3)+q(:,1).*q(:,4));
    R(1,3,:) = 2.*(q(:,2).*q(:,4)-q(:,1).*q(:,3));
    R(2,1,:) = 2.*(q(:,2).*q(:,3)-q(:,1).*q(:,4));
    R(2,2,:) = 2.*q(:,1).^2-1+2.*q(:,3).^2;
    R(2,3,:) = 2.*(q(:,3).*q(:,4)+q(:,1).*q(:,2));
    R(3,1,:) = 2.*(q(:,2).*q(:,4)+q(:,1).*q(:,3));
    R(3,2,:) = 2.*(q(:,3).*q(:,4)-q(:,1).*q(:,2));
    R(3,3,:) = 2.*q(:,1).^2-1+2.*q(:,4).^2;
end
```

3.1.6

```
function qConj = quaternConj(q)
    qConj = [q(:,1) -q(:,2) -q(:,3) -q(:,4)];
end
```

3.1.7

```
function ab = quaternProd(a, b)
    ab(:,1) = a(:,1).*b(:,1)-a(:,2).*b(:,2)-a(:,3).*b(:,3)-a(:,4).*b(:,4);
    ab(:,2) = a(:,1).*b(:,2)+a(:,2).*b(:,1)+a(:,3).*b(:,4)-a(:,4).*b(:,3);
    ab(:,3) = a(:,1).*b(:,3)-a(:,2).*b(:,4)+a(:,3).*b(:,1)+a(:,4).*b(:,2);
    ab(:,4) = a(:,1).*b(:,4)+a(:,2).*b(:,3)-a(:,3).*b(:,2)+a(:,4).*b(:,1);
end
```

3.1.8

```
function v = quaternRotate(v, q)
    [row col] = size(v);
    v0XYZ = quaternProd(quaternProd(q, [zeros(row, 1) v]), quaternConj(q));
    v = v0XYZ(:, 2:4);
end
```

3.1.9

```
function euler = rotMat2euler(R)

    phi = atan2(R(3,2,:), R(3,3,:));
    theta = -atan(R(3,1,:) ./ sqrt(1-R(3,1:2,:).^2));
    psi = atan2(R(2,1,:), R(1,1,:));

    euler = [phi(1,:) theta(1,:) psi(1,:)]';
end
```

3.1.10

```
function q = rotMat2quatern(R)
    [row col numR] = size(R);
    q = zeros(numR, 4);
    K = zeros(4,4);
    for i = 1:numR
        K(1,1) = (1/3) * (R(1,1,i) - R(2,2,i) - R(3,3,i));
        K(1,2) = (1/3) * (R(2,1,i) + R(1,2,i));
        K(1,3) = (1/3) * (R(3,1,i) + R(1,3,i));
        K(1,4) = (1/3) * (R(2,3,i) - R(3,2,i));
        K(2,1) = (1/3) * (R(2,1,i) + R(1,2,i));
        K(2,2) = (1/3) * (R(2,2,i) - R(1,1,i) - R(3,3,i));
        K(2,3) = (1/3) * (R(3,2,i) + R(2,3,i));
        K(2,4) = (1/3) * (R(3,1,i) - R(1,3,i));
        K(3,1) = (1/3) * (R(3,1,i) + R(1,3,i));
        K(3,2) = (1/3) * (R(3,2,i) + R(2,3,i));
        K(3,3) = (1/3) * (R(3,3,i) - R(1,1,i) - R(2,2,i));
        K(3,4) = (1/3) * (R(1,2,i) - R(2,1,i));
        K(4,1) = (1/3) * (R(2,3,i) - R(3,2,i));
        K(4,2) = (1/3) * (R(3,1,i) - R(1,3,i));
        K(4,3) = (1/3) * (R(1,2,i) - R(2,1,i));
        K(4,4) = (1/3) * (R(1,1,i) + R(2,2,i) + R(3,3,i));
        [V,D] = eig(K);
        q(i,:) = V(:,4);
        q(i,:) = [q(i,4) q(i,1) q(i,2) q(i,3)];
    end
end
```

3.2 AHRS (attitude and heading reference system)

```
classdef AHRS < handle
```

Public properties

```
properties (Access = public)
    SamplePeriod = 1/256;
    Quaternion = [1 0 0 0];    % output quaternion describing the sensor relative to the Earth
    Kp = 2;                    % proportional gain
    Ki = 0;                    % integral gain
    KpInit = 200;              % proportional gain used during initialisation
    InitPeriod = 5;            % initialisation period in seconds
end
```

Private properties

```
properties (Access = private)
    q = [1 0 0 0];            % internal quaternion describing the Earth relative to the sensor
    IntError = [0 0 0]';      % integral error
    KpRamped;                  % internal proportional gain used to ramp during initialisation
end
```

Public methods

```
methods (Access = public)
    function obj = AHRS(varargin)
        for i = 1:2:nargin
            if strcmp(varargin{i}, 'SamplePeriod'), obj.SamplePeriod = varargin{i+1};
            elseif strcmp(varargin{i}, 'Quaternion')
                obj.Quaternion = varargin{i+1};
                obj.q = quaternConj(obj.Quaternion);
            elseif strcmp(varargin{i}, 'Kp'), obj.Kp = varargin{i+1};
            elseif strcmp(varargin{i}, 'Ki'), obj.Ki = varargin{i+1};
            elseif strcmp(varargin{i}, 'KpInit'), obj.KpInit = varargin{i+1};
            elseif strcmp(varargin{i}, 'InitPeriod'), obj.InitPeriod = varargin{i+1};
            else error('Invalid argument');
            end
            obj.KpRamped = obj.KpInit;
        end;
    end
    function obj = Update(obj, Gyroscope, Accelerometer, Magnetometer)
        error('This method is unimplemented');
    end
    function obj = UpdateIMU(obj, Gyroscope, Accelerometer)
        % Normalise accelerometer measurement
        if(norm(Accelerometer) == 0) % handle NaN
            warning(0, 'Accelerometer magnitude is zero. Algorithm update aborted.');
```

```
            return;
        else
            Accelerometer = Accelerometer / norm(Accelerometer); % normalise measurement
        end
        % Compute error between estimated and measured direction of gravity
        v = [2*(obj.q(2)*obj.q(4) - obj.q(1)*obj.q(3))
            2*(obj.q(1)*obj.q(2) + obj.q(3)*obj.q(4))
```

```

        obj.q(1)^2 - obj.q(2)^2 - obj.q(3)^2 + obj.q(4)^2];           % estimated direction of gravity
    error = cross(v, Accelerometer');
    obj.IntError = obj.IntError + error;
%
    end
% Apply feedback terms
    Ref = Gyroscope - (obj.Kp*error + obj.Ki*obj.IntError)';
% Compute rate of change of quaternion
    pDot = 0.5 * obj.quaternProd(obj.q, [0 Ref(1) Ref(2) Ref(3)]); % compute rate of change of quaternion
    obj.q = obj.q + pDot * obj.SamplePeriod;                       % integrate rate of change of quaternion
    obj.q = obj.q / norm(obj.q);                                    % normalise quaternion
% Store conjugate
    obj.Quaternion = obj.quaternConj(obj.q);
end
function obj = Reset(obj)
    obj.KpRamped = obj.KpInit;           % start Kp ramp-down
    obj.IntError = [0 0 0]';             % reset integral terms
    obj.q = [1 0 0 0];                   % set quaternion to alignment
end
end

```

Get/set methods

```

methods
function obj = set.Quaternion(obj, value)
    if(norm(value) == 0)
        error('Quaternion magnitude cannot be zero.');
```

```

    end
    value = value / norm(value);
    obj.Quaternion = value;
    obj.q = obj.quaternConj(value);
end
end

```

Private methods

```

methods (Access = private)
function ab = quaternProd(obj, a, b)
    ab(:,1) = a(:,1).*b(:,1)-a(:,2).*b(:,2)-a(:,3).*b(:,3)-a(:,4).*b(:,4);
    ab(:,2) = a(:,1).*b(:,2)+a(:,2).*b(:,1)+a(:,3).*b(:,4)-a(:,4).*b(:,3);
    ab(:,3) = a(:,1).*b(:,3)-a(:,2).*b(:,4)+a(:,3).*b(:,1)+a(:,4).*b(:,2);
    ab(:,4) = a(:,1).*b(:,4)+a(:,2).*b(:,3)-a(:,3).*b(:,2)+a(:,4).*b(:,1);
end
function qConj = quaternConj(obj, q)
    qConj = [q(:,1) -q(:,2) -q(:,3) -q(:,4)];
end
end
end

```


3.3 SixD of Animation

```
function fig = SixDOFanimation(varargin)
```

Create local variables

```
% Required arguments
p = varargin{1};           % position of body
R = varargin{2};           % rotation matrix of body
[numSamples dummy] = size(p);

% Default values of optional arguments
SamplePlotFreq = 1;
Trail = 'off';
LimitRatio = 1;
Position = [];
FullScreen = false;
View = [30 20];
AxisLength = 1;
ShowArrowHead = 'on';
xlabel = 'x';
ylabel = 'y';
zlabel = 'z';
Title = '6DOF Animation';
ShowLegend = true;
CreateAVI = false;
AVIfileName = '6DOF Animation';
AVIfileNameEnum = true;
AVIfps = 30;

for i = 3:2:nargin
    if strcmp(varargin{i}, 'SamplePlotFreq'), SamplePlotFreq = varargin{i+1};
    elseif strcmp(varargin{i}, 'Trail')
        Trail = varargin{i+1};
        if(~strcmp(Trail, 'off') && ~strcmp(Trail, 'DotsOnly') && ~strcmp(Trail, 'All'))
            error('Invalid argument. Trail must be 'off', 'DotsOnly' or 'All'.');
        end
    elseif strcmp(varargin{i}, 'LimitRatio'), LimitRatio = varargin{i+1};
    elseif strcmp(varargin{i}, 'Position'), Position = varargin{i+1};
    elseif strcmp(varargin{i}, 'FullScreen'), FullScreen = varargin{i+1};
    elseif strcmp(varargin{i}, 'View'), View = varargin{i+1};
    elseif strcmp(varargin{i}, 'AxisLength'), AxisLength = varargin{i+1};
    elseif strcmp(varargin{i}, 'ShowArrowHead'), ShowArrowHead = varargin{i+1};
    elseif strcmp(varargin{i}, 'xlabel'), xlabel = varargin{i+1};
    elseif strcmp(varargin{i}, 'ylabel'), ylabel = varargin{i+1};
    elseif strcmp(varargin{i}, 'zlabel'), zlabel = varargin{i+1};
    elseif strcmp(varargin{i}, 'Title'), Title = varargin{i+1};
    elseif strcmp(varargin{i}, 'ShowLegend'), ShowLegend = varargin{i+1};
    elseif strcmp(varargin{i}, 'CreateAVI'), CreateAVI = varargin{i+1};
    elseif strcmp(varargin{i}, 'AVIfileName'), AVIfileName = varargin{i+1};
    elseif strcmp(varargin{i}, 'AVIfileNameEnum'), AVIfileNameEnum = varargin{i+1};
    elseif strcmp(varargin{i}, 'AVIfps'), AVIfps = varargin{i+1};
    else error('Invalid argument. ');
end
end;
```

Reduce data to samples to plot only

```
p = p(1:SamplePlotFreq:numSamples, :);
R = R(:, :, 1:SamplePlotFreq:numSamples) * AxisLength;
if(numel(View) > 2)
    View = View(1:SamplePlotFreq:numSamples, :);
end
[numPlotSamples dummy] = size(p);
```

Setup AVI file

```
aviobj = []; % create null object
if(CreateAVI)
    fileName = strcat(AVIfilename, '.avi');
    if(exist(fileName, 'file'))
        if(AVIfilenameEnum) % if file name exists and
enum enabled
            i = 0;
            while(exist(fileName, 'file')) % find un-used file name by
appending enum
                fileName = strcat(AVIfilename, sprintf('%i', i), '.avi');
                i = i + 1;
            end
        else % else file name exists and enum
disabled
            fileName = []; % file will not be created
        end
    end
    if(isempty(fileName))
        sprintf('AVI file not created as file already exists.')
    else
        aviobj = avifile(fileName, 'fps', AVIfps, 'compression', 'Cinepak', 'quality', 100);
    end
end
```

Setup figure and plot

```
% Create figure
fig = figure('Number', 'off', 'Name', '6DOF Animation');
if(FullScreen)
    screenSize = get(0, 'ScreenSize');
    set(fig, 'Position', [0 0 screenSize(3) screenSize(4)]);
elseif(~isempty(Position))
    set(fig, 'Position', Position);
end
set(gca, 'drawmode', 'fast');
lighting phong;
set(gcf, 'Renderer', 'zbuffer');
hold on;
axis equal;
grid on;
view(View(1, 1), View(1, 2));
title(i);
xlabel(Xlabel);
ylabel(Ylabel);
zlabel(Zlabel);
```

```

% Create plot data arrays
if(strcmp(Trail, 'DotsOnly') || strcmp(Trail, 'All'))
    x = zeros(numPlotSamples, 1);
    y = zeros(numPlotSamples, 1);
    z = zeros(numPlotSamples, 1);
end
if(strcmp(Trail, 'All'))
    ox = zeros(numPlotSamples, 1);
    oy = zeros(numPlotSamples, 1);
    oz = zeros(numPlotSamples, 1);
    ux = zeros(numPlotSamples, 1);
    vx = zeros(numPlotSamples, 1);
    wx = zeros(numPlotSamples, 1);
    uy = zeros(numPlotSamples, 1);
    vy = zeros(numPlotSamples, 1);
    wy = zeros(numPlotSamples, 1);
    uz = zeros(numPlotSamples, 1);
    vz = zeros(numPlotSamples, 1);
    wz = zeros(numPlotSamples, 1);
end
x(1) = p(1,1);
y(1) = p(1,2);
z(1) = p(1,3);
ox(1) = x(1);
oy(1) = y(1);
oz(1) = z(1);
ux(1) = R(1,1,1:1);
vx(1) = R(2,1,1:1);
wx(1) = R(3,1,1:1);
uy(1) = R(1,2,1:1);
vy(1) = R(2,2,1:1);
wy(1) = R(3,2,1:1);
uz(1) = R(1,3,1:1);
vz(1) = R(2,3,1:1);
wz(1) = R(3,3,1:1);

% Create graphics handles
orgHandle = plot3(x, y, z, 'k. ');
if>ShowArrowHead
    ShowArrowHeadStr = 'on';
else
    ShowArrowHeadStr = 'off';
end
quivXhandle = quiver3(ox, oy, oz, ux, vx, wx, 'r', 'ShowArrowHead', ShowArrowHeadStr, 'MaxHeadSize',
0.999999, 'AutoScale', 'off');
quivYhandle = quiver3(ox, oy, oz, uy, vy, wy, 'g', 'ShowArrowHead', ShowArrowHeadStr, 'MaxHeadSize',
0.999999, 'AutoScale', 'off');
quivZhandle = quiver3(ox, ox, oz, uz, vz, wz, 'b', 'ShowArrowHead', ShowArrowHeadStr, 'MaxHeadSize',
0.999999, 'AutoScale', 'off');

% Create legend
if>ShowLegend
    legend('Origin', 'x', 'y', 'z');
end

% Set initial limits
xlim = [x(1)-AxisLength x(1)+AxisLength] * LimitRatio;
ylim = [y(1)-AxisLength y(1)+AxisLength] * LimitRatio;

```

```

Zlim = [z(1)-AxisLength z(1)+AxisLength] * LimitRatio;
set(gca, 'xlim', xlim, 'ylim', ylim, 'zlim', zlim);

```

```

% Set initial view
view(View(1, :));

```

Plot one sample at a time

```

for i = 1:numPlotSamples

    % Update graph title
    if(strcmp(Title, ''))
        titleText = sprintf('Sample %i of %i', 1+((i-1)*SamplePlotFreq), numSamples);
    else
        titleText = strcat(Title, ' (', sprintf('Sample %i of %i', 1+((i-1)*SamplePlotFreq), numSamples),
    ');

    end
    title(titleText);

    % Plot body x y z axes
    if(strcmp(Trail, 'DotsOnly') || strcmp(Trail, 'All'))
        x(1:i) = p(1:i,1);
        y(1:i) = p(1:i,2);
        z(1:i) = p(1:i,3);
    else
        x = p(i,1);
        y = p(i,2);
        z = p(i,3);
    end
    if(strcmp(Trail, 'All'))
        ox(1:i) = p(1:i,1);
        oy(1:i) = p(1:i,2);
        oz(1:i) = p(1:i,3);
        ux(1:i) = R(1,1,1:i);
        vx(1:i) = R(2,1,1:i);
        wx(1:i) = R(3,1,1:i);
        uy(1:i) = R(1,2,1:i);
        vy(1:i) = R(2,2,1:i);
        wy(1:i) = R(3,2,1:i);
        uz(1:i) = R(1,3,1:i);
        vz(1:i) = R(2,3,1:i);
        wz(1:i) = R(3,3,1:i);
    else
        ox = p(i,1);
        oy = p(i,2);
        oz = p(i,3);
        ux = R(1,1,i);
        vx = R(2,1,i);
        wx = R(3,1,i);
        uy = R(1,2,i);
        vy = R(2,2,i);
        wy = R(3,2,i);
        uz = R(1,3,i);
        vz = R(2,3,i);
        wz = R(3,3,i);
    end
    set(orgHandle, 'xdata', x, 'ydata', y, 'zdata', z);
    set(quivXhandle, 'xdata', ox, 'ydata', oy, 'zdata', oz, 'udata', ux, 'vdata', vx, 'wdata', wx);
    set(quivYhandle, 'xdata', ox, 'ydata', oy, 'zdata', oz, 'udata', uy, 'vdata', vy, 'wdata', wy);

```

```

set(quivZhandle, 'xdata', ox, 'ydata', oy, 'zdata', oz, 'udata', uz, 'vdata', vz, 'wdata', wz);

% Adjust axes for snug fit and draw
axisLimChanged = false;
if((p(i,1) - AxisLength) < Xlim(1)), Xlim(1) = p(i,1) - LimitRatio*AxisLength; axisLimChanged = true;
end
if((p(i,2) - AxisLength) < Ylim(1)), Ylim(1) = p(i,2) - LimitRatio*AxisLength; axisLimChanged = true;
end
if((p(i,3) - AxisLength) < Zlim(1)), Zlim(1) = p(i,3) - LimitRatio*AxisLength; axisLimChanged = true;
end
if((p(i,1) + AxisLength) > Xlim(2)), Xlim(2) = p(i,1) + LimitRatio*AxisLength; axisLimChanged = true;
end
if((p(i,2) + AxisLength) > Ylim(2)), Ylim(2) = p(i,2) + LimitRatio*AxisLength; axisLimChanged = true;
end
if((p(i,3) + AxisLength) > Zlim(2)), Zlim(2) = p(i,3) + LimitRatio*AxisLength; axisLimChanged = true;
end

if(axisLimChanged), set(gca, 'Xlim', Xlim, 'Ylim', Ylim, 'Zlim', Zlim); end
drawnow;

% Adjust view
if(numel(View) > 2)
    view(View(i, :));
end

% Add frame to AVI object
if(~isempty(aviobj))
    frame = getframe(fig);
    aviobj = addframe(aviobj, frame);
end

end

hold off;

% Close AVI file
if(~isempty(aviobj))
    aviobj = close(aviobj);
end

end

```

3.4 Script

```
clear;
close all;
clc;
addpath('Quaternions');
addpath('ximu_matlab_library');
% -----
% Select dataset (comment in/out)

filePath = 'Datasets/straightLine';
startTime = 6;
stopTime = 26;

% filePath = 'Datasets/stairsAndCorridor';
% startTime = 5;
% stopTime = 53;

% filePath = 'Datasets/spiralStairs';
% startTime = 4;
% stopTime = 47;
% -----
% Import data

samplePeriod = 1/256;
xIMUdata = xIMUdataClass(filePath, 'InertialMagneticSampleRate', 1/samplePeriod);
time = xIMUdata.CalInertialAndMagneticData.Time;
gyrX = xIMUdata.CalInertialAndMagneticData.Gyroscope.X;
gyrY = xIMUdata.CalInertialAndMagneticData.Gyroscope.Y;
gyrZ = xIMUdata.CalInertialAndMagneticData.Gyroscope.Z;
accX = xIMUdata.CalInertialAndMagneticData.Accelerometer.X;
accY = xIMUdata.CalInertialAndMagneticData.Accelerometer.Y;
accZ = xIMUdata.CalInertialAndMagneticData.Accelerometer.Z;
clear('xIMUdata');
% -----
% Manually frame data

% startTime = 0;
% stopTime = 10;

indexSel = find(sign(time-startTime)+1, 1) : find(sign(time-stopTime)+1, 1);
time = time(indexSel);
gyrX = gyrX(indexSel, :);
gyrY = gyrY(indexSel, :);
gyrZ = gyrZ(indexSel, :);
accX = accX(indexSel, :);
accY = accY(indexSel, :);
accZ = accZ(indexSel, :);
% -----
% Detect stationary periods

% Compute accelerometer magnitude
acc_mag = sqrt(accX.*accX + accY.*accY + accZ.*accZ);

% HP filter accelerometer data
filtCutoff = 0.001;
[b, a] = butter(1, (2*filtCutoff)/(1/samplePeriod), 'high');
acc_magFilt = filtfilt(b, a, acc_mag);
```

```

% Compute absolute value
acc_magFilt = abs(acc_magFilt);

% LP filter accelerometer data
filtCutoff = 5;
[b, a] = butter(1, (2*filtCutoff)/(1/samplePeriod), 'low');
acc_magFilt = filtfilt(b, a, acc_magFilt);

% Threshold detection
stationary = acc_magFilt < 0.05;
% -----
% Plot data raw sensor data and stationary periods

figure('Position', [9 39 900 600], 'Number', 'off', 'Name', 'Sensor Data');
ax(1) = subplot(2,1,1);
    hold on;
    plot(time, gyrX, 'r');
    plot(time, gyrY, 'g');
    plot(time, gyrZ, 'b');
    title('Gyroscope');
    xlabel('Time (s)');
    ylabel('Angular velocity ( $\wedge$ \circ/s)');
    legend('X', 'Y', 'Z');
    hold off;
ax(2) = subplot(2,1,2);
    hold on;
    plot(time, accX, 'r');
    plot(time, accY, 'g');
    plot(time, accZ, 'b');
    plot(time, acc_magFilt, ':k');
    plot(time, stationary, 'k', 'Linewidth', 2);
    title('Accelerometer');
    xlabel('Time (s)');
    ylabel('Acceleration (g)');
    legend('X', 'Y', 'Z', 'Filtered', 'Stationary');
    hold off;
linkaxes(ax,'x');
% -----
% Compute orientation

quat = zeros(length(time), 4);
AHRSalgorithm = AHRS('SamplePeriod', 1/256, 'Kp', 1, 'KpInit', 1);

% Initial convergence
initPeriod = 2;
indexSel = 1 : find(sign(time-(time(1)+initPeriod))+1, 1);
for i = 1:2000
    AHRSalgorithm.UpdateIMU([0 0 0], [mean(accX(indexSel)) mean(accY(indexSel)) mean(accZ(indexSel))]);
end

% For all data
for t = 1:length(time)
    if(stationary(t))
        AHRSalgorithm.Kp = 0.5;
    else
        AHRSalgorithm.Kp = 0;
    end
    AHRSalgorithm.UpdateIMU(deg2rad([gyrX(t) gyrY(t) gyrZ(t)]), [accX(t) accY(t) accZ(t)]);

```

```

    quat(t,:) = AHRSalgorithm.Quaternion;
end
% -----
% Compute translational accelerations

% Rotate body accelerations to Earth frame
acc = quaternRotate([accX accY accZ], quaternConj(quat));

% % Remove gravity from measurements
% acc = acc - [zeros(length(time), 2) ones(length(time), 1)];    % unnecessary due to velocity integral drift
% compensation

% Convert acceleration measurements to m/s/s
acc = acc * 9.81;

% Plot translational accelerations
figure('Position', [9 39 900 300], 'Number', 'off', 'Name', 'Accelerations');
hold on;
plot(time, acc(:,1), 'r');
plot(time, acc(:,2), 'g');
plot(time, acc(:,3), 'b');
title('Acceleration');
xlabel('Time (s)');
ylabel('Acceleration (m/s/s)');
legend('x', 'y', 'z');
hold off;
% -----
% Compute translational velocities

acc(:,3) = acc(:,3) - 9.81;

% Integrate acceleration to yield velocity
vel = zeros(size(acc));
for t = 2:length(vel)
    vel(t,:) = vel(t-1,:) + acc(t,:) * samplePeriod;
    if(stationary(t) == 1)
        vel(t,:) = [0 0 0];    % force zero velocity when foot stationary
    end
end

% Compute integral drift during non-stationary periods
velDrift = zeros(size(vel));
stationaryStart = find([0; diff(stationary)] == -1);
stationaryEnd = find([0; diff(stationary)] == 1);
for i = 1:numel(stationaryEnd)
    driftRate = vel(stationaryEnd(i)-1, :) / (stationaryEnd(i) - stationaryStart(i));
    enum = 1:(stationaryEnd(i) - stationaryStart(i));
    drift = [enum'*driftRate(1) enum'*driftRate(2) enum'*driftRate(3)];
    velDrift(stationaryStart(i):stationaryEnd(i)-1, :) = drift;
end

% Remove integral drift
vel = vel - velDrift;

% Plot translational velocity
figure('Position', [9 39 900 300], 'Number', 'off', 'Name', 'velocity');
hold on;
plot(time, vel(:,1), 'r');
plot(time, vel(:,2), 'g');

```



```

plot(time, vel(:,3), 'b');
title('Velocity');
xlabel('Time (s)');
ylabel('velocity (m/s)');
legend('x', 'y', 'z');
hold off;
% -----
% Compute translational position

% Integrate velocity to yield position
pos = zeros(size(vel));
for t = 2:length(pos)
    pos(t,:) = pos(t-1,:) + vel(t,:) * samplePeriod;    % integrate velocity to yield position
end

% Plot translational position
figure('Position', [9 39 900 600], 'Number', 'off', 'Name', 'Position');
hold on;
plot(time, pos(:,1), 'r');
plot(time, pos(:,2), 'g');
plot(time, pos(:,3), 'b');
title('Position');
xlabel('Time (s)');
ylabel('Position (m)');
legend('x', 'y', 'z');
hold off;
% -----
% Plot 3D foot trajectory

% % Remove stationary periods from data to plot
% posPlot = pos(find(~stationary), :);
% quatPlot = quat(find(~stationary), :);
posPlot = pos;
quatPlot = quat;

% Extend final sample to delay end of animation
extraTime = 20;
onesVector = ones(extraTime*(1/samplePeriod), 1);
posPlot = [posPlot; [posPlot(end, 1)*onesVector, posPlot(end, 2)*onesVector, posPlot(end, 3)*onesVector]];
quatPlot = [quatPlot; [quatPlot(end, 1)*onesVector, quatPlot(end, 2)*onesVector, quatPlot(end, 3)*onesVector,
    quatPlot(end, 4)*onesVector]];

% Create 6 DOF animation
SamplePlotFreq = 4;
Spin = 120;
SixDofAnimation(posPlot, quatern2rotMat(quatPlot), ...
    'SamplePlotFreq', SamplePlotFreq, 'Trail', 'All', ...
    'Position', [9 39 1280 768], 'view', [(100:(Spin/(length(posPlot)-1)):(100+Spin))',
10*ones(length(posPlot), 1)], ...
    'AxisLength', 0.1, 'ShowArrowHead', false, ...
    'xlabel', 'X (m)', 'ylabel', 'Y (m)', 'zlabel', 'Z (m)', 'showLegend', false, ...
    'CreateAVI', false, 'AVIfileNameEnum', false, 'AVIfps', ((1/samplePeriod) / SamplePlotFreq));

```

4 Sensors

All sensors will be controlled and managed by the RASPBERRY PI 2 MODEL B.

4.1 Color Sensor

Color Detector Sensor “<https://www.sparkfun.com/products/retired/11195>”, as it was mentioned earlier that it is used as one on each side of the pod to determine the exact location of the pod inside the tube and to determine when to activate the braking system.

4.2 Temperature Sensor

Grove - High Temperature Sensor “<http://www.trossenrobotics.com/grove-high-temp-sensor>”, it uses a K type thermocouple for temperature detection, with a Thermistor to detect the ambient temperature, used as a cold-compensation reference. The detectable range of this Sensor is -50°C to 600°C (-58°F to 1112°F), and the accuracy is rated at $\pm(2.0\% + 2^{\circ}\text{C})$.

This sensor is used for tracking the thermal profile of the pod inside.

4.3 Gas Pressure Sensor

Phidgets Differential Gas Pressure Sensor “<http://www.trossenrobotics.com/p/phidget-differential-gas-pressure-sensor.aspx>”, it is a dual ported gas pressure sensor that measures the pressure difference between the two ports. It can measure both positive and negative pressure from -25kPa to +25 kPa.

It is used to track and determine the air pressure inside the tube to make sure that it is the optimum operating air pressure for the designed pod, and it could be used inside the pod to measure pressure variations inside the pod to make sure that it is breathable for onboard humans.

4.4 Slam Stick X-Metal

It is the Slam Stick X – Metal “<http://www.mide.com/collections/shock-vibration-data-loggers/products/slam-stick-x-metal>” provided by SpaceX to be installed on pod. The Slam Stick X incorporates a tri-axial accelerometer, temperature sensor, pressure sensor, and data acquisition system into one, easy-to-use product. Its ability to measure a wide range of acceleration amplitude, high frequency sampling rates (up to 20 kHz), long battery life (over 15 hours), large memory (up to 4 billion samples), and small, portable form factor make it ideal for many vibration measurement and impact testing applications. The Slam Stick X also now comes with the option to include a second DC response accelerometer for improved shock and vibration testing. Vibration analysis is made easy with the free Slam Stick Lab software.

5 Power

The total required power is almost 6W with a range of voltages between 3.1V – 6.3V and a range of currents between 50mA – 150 mA, excluding the Slam Stick X-Metal as it has its own battery with a long life of (over 15 hours). So we chose this Polymer Li-Ion battery: 3.7V 210mAh (0.78Wh, 0.21A rate) with ZMR-3 Connector - UN38.3 Passed “<http://www.batteryspace.com/Polymer-Li-Ion-battery-3.7V-210mAh-with-ZMR-3.aspx>” to power all the electronic components required for navigating the pod and tracking its systems.

6 Cost

#	Item	Price	Quantity	Total Price
1	x-IMU	\$280	1	\$280
2	RASPBERRY PI 2 MODEL B 8GB	\$40	1	\$40
3	Color Detector Sensor	\$53	2	\$106
4	High Temperature Sensor	\$9	2	\$18
5	Phidgets Differential Gas Pressure Sensor	\$32	2	\$64
6	Slam Stick X - Metal	\$1750	1	\$1750
7	Polymer Li-Ion battery: 3.7V 210mAh	\$20	2	\$40
				\$2298

The total cost of this system is \$2298

7 Maintenance

The maintenance of this system is simple as you can easily replace any device having a failure as it does not cost a lot. If it has a failure like a broken pin or disconnected wire, you can easily reconnect or weld them. Otherwise, you will have to replace the device.



Nova Hyperloop Team

Faculty of Engineering

Cairo University

Egypt

- **Team Contact Information**

1. **Prof. Basman El hadidy**

belhadid@eng.cu.edu.eg

2. **Ahmed Mohamed Mohamed**

+20 114 112 5533

ahmedmohamedsayed94@gmail.com

<https://eg.linkedin.com/in/ahmedahmed94>

3. **Mohamed Hassan Mohamed**

+20 122 989 9411

008mohamed@gmail.com

<https://eg.linkedin.com/in/mohamed-hassan-001b61a3>

4. **Samar Fathy Eldiary**

+20 101 854 3630

samar.eldiary@gmail.com

<https://eg.linkedin.com/pub/samar-eldiary/6b/961/400>

5. **Ahmed Emad Hodaib**

+20 115 167 7886

ahmedhodaib@hotmail.com

<http://eg.linkedin.com/in/ahmedhodaib>